Middlesex University London

Middlesex University Research Repository

An open access repository of

Middlesex University research

http://eprints.mdx.ac.uk

Lee, Chun (2008) Art unlimited: an investigation into contemporary digital arts and the free software movement. PhD thesis, Middlesex University. [Thesis]

Final accepted version (with author's formatting)

This version is available at: https://eprints.mdx.ac.uk/4963/

Copyright:

Middlesex University Research Repository makes the University's research available electronically.

Copyright and moral rights to this work are retained by the author and/or other copyright owners unless otherwise stated. The work is supplied on the understanding that any use for commercial gain is strictly forbidden. A copy may be downloaded for personal, non-commercial, research or study without prior permission and without charge.

Works, including theses and research projects, may not be reproduced in any format or medium, or extensive quotations taken from them, or their content changed in any way, without first obtaining permission in writing from the copyright holder(s). They may not be sold or exploited commercially in any format or medium without the prior written permission of the copyright holder(s).

Full bibliographic details must be given when referring to, or quoting from full items including the author's name, the title of the work, publication details where relevant (place, publisher, date), pagination, and for theses or dissertations the awarding institution, the degree type awarded, and the date of the award.

If you believe that any material held in the repository infringes copyright law, please contact the Repository Team at Middlesex University via the following email address:

eprints@mdx.ac.uk

The item will be removed from the repository while any claim is being investigated.

See also repository copyright: re-use policy: http://eprints.mdx.ac.uk/policies.html#copy

Art Unlimited:
An investigation into contemporary digital arts and the free software movement

A thesis submitted to Middlesex University in partial fulfillment of the requirements for the degree of Doctor of Philosophy/Master of Philosophy

Chun Lee

School of Art
Middlesex University

May, 2008

Abstract

Computing technology has not only significantly shaped many of the contemporary artistic disciplines, it has also given birth to many new and exciting practices. Modest, low cost hardware enabled artists to manipulate real-time multimedia data and coordinate vast amounts of hardware devices, whilst high bandwidth Internet connections has allowed them to communicate and distribute their work rapidly. For this reason, art practices in the digital domain have become highly decentralized.

It is therefore not surprising that the rise of free and open source software (FLOSS) has been warmly welcomed and adopted by an increasing number of practitioners. The technical advantages in free software allows them to create works of art with greater freedom and flexibility. Its open and collaborative ideology, on the other hand, further embraces the increasingly autonomous and distributed characteristics in the artistic community.

This thesis aims to examine the impact of free and open source software in the context of contemporary digital arts. It will look at the current climate of both digital arts and the FLOSS movement, attempting to rationalize the implications of such a phenomena. It will also provide concrete examples of ongoing activities in FLOSS digital arts, as an evidence and documentation of its development to date. Lastly, the practical work in this research will offer a first hand insight into developing a FLOSS project within the given context.

Acknowledgments

I would like to thank Dr. John Dack for the continuous support throughout this research. Without his invaluable input and encouragement, its completion would not have have been possible. I would also like to thank professor Huw Jones for his initial supervision.

I would like to express my gratitude to Inna for her enduring support, as well as my close friends and colleagues. I wish to thank my collaborator, Mathieu Bouchard, for sharing his expertise and knowledge. They have given me the inspiration and motivation to carry out my work.

Lastly, a very special thanks to my parents, Y.C and Grace Lee, to whom I dedicate this thesis.

Contents

List of Figures

Chapter 1

Introduction

1.1 Research Context

Mass production has made technology remarkably accessible to artists. Society's vast demand for electronic devices, consumer or industrial, means that cutting edge designs are being constantly and rapidly developed[1]. As a result, costs of computing power dramatically decrease as hardware becomes evermore capable and abundant. Using technology, or more precisely computers, as a creative medium is now an attractive and common practice amongst contemporary artists. Computer generated works of art or designs are now seen in almost every social scenario, ranging from mass media production to the experimental art scene.

Several factors have contributed to the widespread use of technology based artistic practices. The affordable price of new and modest computers can obviously

[1]Trade shows such as CES (http://www.cesweb.org/default.asp) and CeBIT (http://www.cebit.de) are prime examples where cutting edge designs are introduced to the consumer electronic market. Technologies such as display systems, mass storage devices, high definition video formats and inexpensive, ultra portable laptop computers are some of the areas where advances have been made in recent years. For example, practical applications of OLED (Organic light-emitting diode), Electronic paper, SSD memory are some of the technologies that have been seen in these events.

account for such phenomena. What used to be expensive and impractical, is now an attainable reality for most artists with slender financial means. Further, as ever more hardware is simply disposed of by consumers keeping up with latest trends, they remain functional and can thus be recycled. These "outdated" hardware devices are often offered cost-free, making them a viable supply for artists working with technology. In fact, artistic creations based on recycled or found hardware, have become a dominant practice in contemporary media art, expressing various ethical and sociological concerns. Ready-made computers aside, components such as micro-controllers are also widely available and can be easily customized for specific needs. Whilst a different level of expertise is required to be proficient with technology (depending on the source and type of hardware), this nevertheless illustrates that artists of all abilities are able to work comfortably with computers and produce works of art.

Artistic concerns have also evolved in correlation with the increasing capability and availability of computing hardware. Real-time media processing, artificial life[2] and distributed systems[3] are just a few examples which have become widely popular in the current practices of digital art. The technological possibilities are now immense in comparison with a few decades ago. The shift of creative interests can be clearly observed in the practice of generative arts in the digital domain. No longer satisfied with works that have a static appearance or predetermined structure, artists are now able to construct algorithms which actively take part in the creative process with or without human intervention. In other words, instead of producing recognizable artifacts, artists define "creative" procedures in the forms of computer codes, which in turn generate the final result. Moreover, these system can be constructed rapidly to provide fast and immediate feedback, which enable artists to efficiently achieve the desired design. They can even be employed in the performance context. Several interesting issues may arise in this mode of practice: authorship of the end result, machine creativity[4] and related aesthetics, are common debates amongst its

[2]One well known example is Karl Sims[35] research in applying evolutionary algorithms to generate computer graphics and animations

[3]Electric Sheep, by Scott Draves[12], is an award winning project utilising distributed computing techniques as well as genetic algorithms to render fractal based computer animations

[4]Researching into the nature of creativity has attracted extensive interest in both art and

practitioners. Other forms of digital arts that are equally exciting and challenging also exist, exploring various aspects of digital art as a result of technological advances.

Computers need software to be useful and functional. For this reason, the widespread use of computing hardware equally highlights the significance of software and its implications. Software is now also in abundance, fulfilling creative tasks of all types and scales. Software can be bought, downloaded, exchanged and even modified at the artists' will to suite their individual circumstances. Furthermore, programming is becoming evermore accessible to artists, with many computer languages specifically designed for creative purposes and easy to learn. Because of this, the ways in which software tools are obtained or created can have a very direct impact on a given artistic practice. For instance, given the same hardware, artists who employ commercially packaged programs and those who develops their own software would often have contrasting ideology and aesthetics.

Despite the correlation between computer hardware and software, several fundamental differences do exist and set them apart. Computing hardware is the apparatus which, when applied appropriately, becomes useful. These devices, therefore, are analogical to any other physical objects invented to assist all kinds of tasks. Manufacturing them, therefore, involves traditional industrial processes, from gathering material to producing required components and the final assembly. Software programs, on the other hand, are essentially logical instructions which command the corresponding hardware. The nature of software therefore, is similar to abstract knowledge rather than to that of a concrete man-made artifact. For instance, software is often compared to recipes, which can be learned and passed on without a physical medium, and when executed correctly, produce pleasurable dishes[5]. The production of software essentially

science. The ultimate goal of these investigations, to a large extent, is to methodologically understand the underlying mechanisms in various types of creative processes, thus to effectively model them using computer systems. For instance, Martin Dostàl's 2005 paper attempted to model musical creativity through generating human-like rhythmic accompaniment[11]. Notable writings on the subject of creativity and its artificial modeling includes Bohm's 'On creativity'[5], Boden's 'The creative mind'[4] and Partridge's 'Computers and creativity'[27].

[5]Such a comparison between software and recipes were often used by Richard Stallman,

involves programmers writing computer codes which are then to be duplicated and released. Physical packaging aside, the nature of software is virtual.

Some effects resulting from this fundamental difference rapidly emerge. Because of the manufacturing process, the more a particular device has been produced, the cheaper it usually becomes. However, this does not apply to software programs, as the scale of production has no real impact on the final cost. On the contrary, the price for software often increases according to the demand. Furthermore, users cannot easily change or replace the physical design of a given piece of hardware; they are however able to try and use different types of software on the same computer. In other words, it can be argued that users have more influence over the software they use, than on the hardware they own.

From an artistic perspective, the role software plays can be more critical and complex. For some artists, software programs are essentially means of production, just like the hardware available to them. For others, software encompasses a higher level of creative, and even ethical, beliefs. Their concerns transcend individual modes of practice and focus on issues such as freedoms of expression and the dissemination of technology. Moreover, the evolution of software is even having an impact on the way hardware is currently being produced.

Being practice-based, the context of this research is also based on personal experience as an ongoing creative practitioner. Starting in early 2000, the practice began with the focus on various generative compositional techniques in live performance. In particular, the investigation into genetic algorithms as means of producing rhythmic structures[6] constituted the previous academic study, and resulted in a series of purpose built, customized software tools[7]. Since then, it developed to take on a much broader interest in software and algorithmic composition in general. Whilst the emphasis on live performance still remains, it also started to actively relate to and work alongside other artists in the field. The efforts led to fruitful collaborations, ranging from event organization to soft-

founder of the Free Software Foundation.

[6]The basis and main influences of this investigation includes 'The Blind Watchmaker'[10] by Richard Dawkins and 'Emergence'[16] by the creator of genetic algorithms John H. Holland.

[7]A series Max/Msp objects was developed to explore the use of Genetic Algorithms as compositional technique.

ware development. Earlier interests in generative systems also extended into the domain of software development and distribution. To be more precise, it finds parallels between the open compositional approach and the ideologies of the free software movement. In both scenarios, the end result, works of art or software programs, is ever evolving according to the rules and conventions set by the environment. The fascination for the free software culture has since become a prominent part of the current practice. For instance, every aspect of both the thesis and the practical project are entirely produced using free software tools[8].

It is worth pointing out that this research does not draw particular distinctions between different types and forms of digital arts. In other words, although the personal artistic background is derived from algorithmic composition in sonic arts, the context of this research will also apply to other creative disciplines, such as visual arts. The intention is to consider digital arts in its own right, as it has the ability to transcend the categorical division between conventional genres of practices.

1.2 Research Motives

The rich context found in contemporary digital arts thus cultivates a diverse range of different practices. As mentioned previously, generative art has brought interesting issues and challenges to the current understanding of arts. It highlights and ultimately attempts to externalize the notion of creativity in the form of formal systems. Software art, another dominant artistic movement, focuses on the aesthetics of computer codes and the abstract processes they govern. The process of programming and the social impact of software are some other examples tackled by other types of practices. This research, as a result, is largely motivated by the wealth of ongoing activities found in the field of digital arts.

Artistic concerns aside, the sociological aspects of digital arts are also experiencing critical transformation. With high bandwidth and wireless Internet connection, artists are constantly seeking ways to exchange information and even collaborate in cyberspace. As a result, virtual alliances are frequently

[8]See appendix A for a list of software used throughout this research

formed by artists sharing similar ideologies to better promote and express their views. The relationship between individuals and the collectives are thus highly intricate. Furthermore, the implication of technology-enabled social networks and activities are also an emerging interest amongst the mass media and society in general. An increasing number of virtual platforms have become extremely popular [9] not only for artists but also within the general public, to share a variety of digital content and exchange opinions. The motivation of this research is thus also derived from witnessing the open and collaborative characteristics found amongst these artists.

Conducting this research offers an unique opportunity to contextualize the underlying personal artistic practice. The methodological approach to contemporary digital arts would allow the practice to objectively position itself in the field. This is critical in evaluating the given practice for future development. Moreover, it would also enable the practice to further establish itself through the outcome of the research. Such abilities to broaden the horizons of the personal creative practice is another primary motivation behind the investigation.

In short, the research is motivated by the following points:

- The popularization of digital arts, encouraging the emergence of a wide spectrum of creative activities.
- The conceptual and practical challenges of contemporary digital arts, especially in relation to generative art.
- The underlying social principle and structure of contemporary digital arts.
- Further development of personal compositional/performance practice.

[9] Social network software has become the latest phenomenon since the emergence of web2.0. These web sites enable users to customize their on-line identify by allowing them to personalize its content. They also provide highly targeted communication, so communities of very specific focus can be established. Myspace and FaceBook are two of the most popular sites of this type to date. Although many artists have utilized these tools to promote and establish their practices, the corporate driven and the commercial aspects behind these sites have nevertheless raised serious concerns for others. For them, social networks are better achieved through independently hosted servers that provides the necessary services.

1.3 Research Subjects

Contemporary digital arts, generative art and free software are consequently the three main subject areas of this research. It will examine each of the them individually, as well as analyze the explicit and the implicit relationships they may have. To establish a clear overall outline of the investigation, these subjects will now be briefly introduced. Furthermore, this section also intends to identify how they will be examined, as well as the roles they play in the research.

1.3.1 Digital arts

As making music was one of the first suggested applications for computers when they were first invented[10], it is not surprising that artists responded quickly to the rise of this powerful instrument. Early pioneers and researchers in computer generated/assisted artworks can be traced back to the 1950s and the 1960s[11]. However, the computational power available at that time was limited, and the price for this technology was high, which meant that the digital arts community was mainly restricted to academic institutions and advanced scientific laboratories. Today, however, most modest computers at present have the ability to process real-time media such as sound and visuals. Computers as compositional tools have become truly affordable and practical. Digital arts have indeed come a long way since their very beginning. They have been transformed from being an inaccessible form of art, to creative activities that can be enjoyed by mass public. In other words, anyone who is familiar with computers can use them to produce graphics, music and films with only little effort involved.

Computers as compositional/performance tools are truly unique in many ways, mostly because of their programmable nature and high speed execution capa-

[10] In various written response to the analytical engine (invention of Charles Babbage, it is regarded as the first mechanical computer) between 1842-1843, Augusta Ada King, Countess of Lovelace, suggested "the Engine might compose elaborate and scientific pieces of music of any degree of complexity or extent."[38]

[11] For example, the Illiac Suite by Lejaren Hiller[15] composed in 1957 is often considered to be the first computer generated music. Hiller used the computer to produce music in the form of music score, which was performed by a string quartet

bility. Artists are now able to simulate extremely complex compositional systems which were previously unattainable[12]. Unlike conventional musical instruments which directly respond to human input in a passive but subtly complex way, computers can be programmed to take on an active/interactive role in the process of composition or performance. Computers also provide an invaluable common platform for digital artists and enthusiasts of different disciplines to collaborate. They further enable artists to break free from the boundary set by conventional modes of practice. As a result, new possibilities and ways of thinking can emerge and be developed.

With computer technology becoming increasingly miniaturized and high-speed wireless Internet connection becoming widely available, digital artists are now experiencing further liberation of the virtual from the physical. Artists are now more independent than ever before, but at the same time better connected as a collective. Ideas are formed, transmitted and developed globally in no time. Digital artists are redefining many aspects of human creative practices.

Given its popularity, digital arts as a creative discipline nowadays encompasses a wide variety of forms and activities. Software art, algorithmic art, Internet art and new media art are just a few examples. Moreover, the term is also sometimes used in the mainstream media to describe creative designs and artifacts generally produced with the aid of computer software. Despite the widespread acknowledgment, what the discipline entails may become ill-defined, and can thus lead to misinterpretation or misuse. This is problematic from this research's stand point. For this reason, the investigation will firstly aim to categorically dissect what digital art might mean in today's circumstances. By doing so, the artistic principle of this research can be clearly defined against the common modes of practice. The investigation will then continue to focus on certain types of practice which identifies with the ideology of the research.

[12]Composer and author Trevor Wishart expressed similar notion in the opening chapter of his 1996 book 'On Sonic Art'[40], in which he said he *"... discovered an instrument - the computer - through which I could realize some of the concepts of musical transformation I had been dreaming of for some years"*. Moreover, It can be argued that the basis of 'On Sonic Art' rests upon the artistic concerns and new compositional possibilities brought about by computers.

1.3.2 Generative art

The term 'generative music' was thought to be first coined by artist Brian Eno[13], although the practice of generative art itself can be traced as far back as the self-sounding string instrument[13] found in ancient Greece and China[14]. The popularization of adopting computer programming to create works of art in recent times, has brought the practice of generative arts into a new era in the virtual domain. The following two definitions[15] provide a brief insight into the meaning of generative art.

> "*Generative art is a term given to work which stems from concentrating on the processes involved in producing an artwork, usually (although not strictly) automated by the use of a machine or computer, or by using mathematic or pragmatic instructions to define the rules by which such artworks are executed*" - Adrian Ward[16]

> "*Generative art refers to any art practice where the artist creates a process, such as a set of natural language rules, a computer program, a machine, or other procedural invention, which is then set into motion with some degree of autonomy contributing to or resulting in a completed work of art*" - Philip Galanter[17]

Throughout history, artists of all disciplines typically focused on their creative skills to produce finite works. In other words, once a work is produced, it is

[13] Named Aeolian harp, it was first described in 'Musurgia Universalis'[18] by Athanasius Kircher in Rome, 1650.

[14] Similar to the Aeolian harp, Feng-Zeng is a self sounding string instrument, is essentially a kite fixed with bamboo or silk strips that produce sounds once the kite is in the air. Moreover, pipes are also used to fix onto the kite instead of strips of bamboo. This variety of Feng-Zheng produces tones that resemble the sound of flutes.

[15] These definitions should not be considered as definitive, as the practice is still rapidly evolving. In other words, they will be subject to further refinement when the body of works in the field reaches critical mass.

[16] http://www.generative.net/read/definitions. Note, as of the submission of this thesis (October 2007), all URL referenced throughout are valid.

[17] "What is Generative Art? Complexity Theory as Context for Art Theory", proceedings of International Conference of Generative Art 2003, p.216. [14]

static and fixed in time and space - a traditional music score, a painting, a sculpture or a book are just some examples. However, this notion does not apply in the practice of generative art. Whilst it is true that many modern art movements have placed increasing emphasis on art works which have open forms[18], such types of practices are still relatively new in comparison with the long standing western art tradition.

In generative art, instead of focusing on the final outcome, artists are more concerned and interested by the processes in which aesthetically pleasing forms and structures can be produced. As a result, works in generative arts usually have no fixed representation, they very often change and evolve through time or (virtual) space.

This shift in the focus of creating works of arts is significant. Because of this change of interests, artists are now entering the realm of 'meta-art', where the aim is to explore procedures and rules that in turn create the final outcome. As a result, the practice of generative arts may allow us to gain new insights into arts and creativity [19], as well as developing new kinds of aesthetics.

As one of the most dominant practices in contemporary digital arts, this research will examine its fundamental notions and characteristics. Furthermore, concrete examples of techniques will be presented and discussed, demonstrating the details in the implementation of artistic generative systems. Lastly, the investigation will attempt to highlight the problematic areas generative art is currently facing, and point out possible future developments.

1.3.3 Free software

In recent years, free/libre/open source software (FLOSS) has gained increasing recognition in the public eye as well as in the digital arts community. The

[18]Fluxus art in the 1960s and 1970s is perhaps the most well known example, where the observable outcome of a given art work can change significantly according to external factors such as the surrounding environment, the intention/interpretation of the performer/audience.

[19]The notion of creativity could also be extended into the realm of the machine, as the process of creation is abstracted into quantifiable instructions. This will no doubt have an impact on the traditional humanistic approach to the understanding of art.

fundamental ideology of open source is that software should be free, just like freedom of expression. In other words, software should be freely available for distribution and modification. Such an approach is in many ways the opposite of the conventional model of software development under copyright restriction.

Two main reasons have contributed to the popularization of using free software in the digital arts community, which is firstly freedom of expression and secondly the cost for value. By using free software, artistic conceptions are not bound to the predefined features of a given software, but to the artists' creative ability to utilize and create software tools. In other words, any conceivable designs are made possible by either using existing software packages, or modifying them to suit a particular purpose. The freedom to customize programs based on others' work significantly enables immense technical possibilities for artists. Such a degree of flexibility in software re-use and modification is non-existent and strictly prohibited with proprietary software. Furthermore, as all free software can be legally obtained and redistributed with zero costs, this brings tremendous advantages for artists who have limited financial means. As one of the strengths in free software lies in the support for legacy hardware, this also means that artists are able to achieve complex installations and setups without specialized, and often high-cost equipment.

Moreover, free software is typically developed and maintained voluntarily by user groups, spread across the world and connected by the Internet. Such a model of development typically results in much more stable and innovative programs [20] than the proprietary ones. This is due to the fact that every member in the community of a given software can look for faults in the program and potentially fix or report them, and since many users and developers keep in contact via the Internet, repairs to faults or new features are released rapidly. Furthermore, having such a globally connected user community also means that getting help with particular problems presents few difficulties.

Governed by very few and simple rules of collaboration, the free software com-

[20]The Apache web server (http://apache.org/) and the Linux kernel (http://kernel.org/) are prime examples of the stability and innovations resulted from the open source software development model.

munity is able to produce a high quality, innovative and complex world of open source computer programs. For example, the Apache web server accounts for 69 percents of all web servers on the Internet. Furthermore, Google, the search engine, relies on an exclusively open source (GNU/Linux) operating system to manage around 6000 servers that keeps the search engine running smoothly. The key to the success of free software is that it provides a decentralized, open-ended system with increasing returns, in which computer programs can evolve. In other words, the more popular a given software becomes, the more users and developers it will attract. This community effect can thus fuel the future development of a given project.

In this research, the historical account of the free and open source movements will be provided, alongside some of the latest current developments. It will then investigate the underlying philosophy and ideology behind the movements, pointing out any potential similarities between FLOSS as a technological phenomenon and digital arts as a creative discipline. The impact of FLOSS on contemporary digital art is also one of the main concerns in the investigation, as it will aim to provide concrete evidence of both the sociological and the technological influences FLOSS has on the artists.

1.4 Subject relationship

The theoretical aspect of this research consists of two fundamental elements, which are divided into the conceptual and social components. The purpose of the conceptual framework is to deal with the relevant artistic issues. The social framework, on the other hand, aims to give practical and technical perspectives. They roughly correlate to each of the research subjects previously identified.

The investigation into contemporary digital arts thus constitutes the conceptual and the artistic framework, whilst the study of FLOSS forms the technical and social framework. Digital arts, as a result, essentially define the starting point and the abstract foundation in which rest of the research can be built on. FLOSS, therefore, will demonstrate how the conceptual framework may be materialized and the social impact of such implementation.

The analysis into generative art does not directly fall into either of the two theoretical components, it serves a different purpose. Its aim is to provide tangible examples and thus gain greater understanding into the current development of digital arts. As a result, the findings in the practice of generative art would hopefully contribute to the substance and realistic perspective over the conceptual framework. Generative art aside, other influential practices will also be mentioned to achieve a broader and truthful representation of the artistic understanding.

The relationship between the two frameworks are in many ways complementary. Although the new possibilities and challenges brought by computing technology are clear, without a reliable software platform, they simply cannot be effectively expressed. In other words, without suitable tools, artists are not able to convey their creativity, no matter how innovative they are. FLOSS, as result, delivers a real solution to address this issue, precisely because free software ensures the free flow of GPL[21] licensed programs and their source codes. This instantly gives artists complete access to the ability to create, without any restriction. Through digital art, FLOSS can find yet another welcoming habitat in which it can evolve and flourish.

There are also more subtle relationships between each of the research subjects. Initially, there seems to be very little correlation between generative art and the free software movement, simply because the former is a creative discipline and the latter is a philosophy and a model of software development. However, with a closer look, one can observe the implicit connection from a different perspective. Although many formal procedures can be used to produce aesthetically pleasing results, one particular type seems to have captured the attention and the imagination of many artists. This type of process is defined and governed by very few rules, however, the result produced by it far exceeds the initial simplicity and are often extremely complex. For example, natural evolution is dictated by 'the survival of the fittest', and yet is capable of producing the world as we live in today. The free software community can also be seen as a prime example. Based on simple rules of mutual collaboration, the open source community is a

[21] GPL stands for "GNU General Public License" is the most popular free software license written by Richard Stallman for the GNU project. http://www.gnu.org/

generative system where the outcome is a universe of computer software that evolves rapidly. Because of this, the abstract model of the collective social behavior can hold great interest from generative art's point of view, and is not to be overlooked.

Although there is a long history to the practice of generative art, it still remains one of the less known creative disciplines in general. This is mostly due to the fact that there has never been an efficient medium to express and explore its full potential. Computers, as a result, become a natural habitat for artists to create and manipulate all kinds of creative systems. Through the adoption of digital technology, generative art has become far more practical and accessible.

In short, the three main subject areas remain deeply intertwined throughout the investigation. Whilst FLOSS serves as a practical platform and a social framework, digital arts will provide this research with a medium and conceptual outline. Lastly, generative art and other active practices, will give substance to the investigation.

1.5 Documentation

Besides examining the proposed subjects, this research also aims to capture the ongoing development of FLOSS digital arts in Western Europe. Although the recognition of FLOSS digital art has increased rapidly in recent years, being a young and emerging movement, many aspects of it are yet to be documented. Such documentation can thus be invaluable to the whole artistic community, not only in the view of future social anthropological reference, but also as an acknowledgment to the achievements of the collective to date. The documentation will be divided into three main sections: software tools, artistic collectives and public events. The focus of each section will now be briefly summarized.

By understanding the tools artists use, one can gain greater insight into both their practice and artistic ideology. Following this notion, some of the most prominent and innovative software tools will be included in this section. It will also attempt to present them in a categorical manner, so their functions in the

creative process can be clearly understood. For instance, some software can be seen as having the role of "creative environments" whereas others are inclined towards being useful utilities. The former allows artists to construct their works of art whilst the latter provides effective practical solutions to solve specific task.

The documentation will then reflect on a variety of FLOSS artistic collectives. An overview will be given to reveal the wide ranging approaches and characteristics between them. Furthermore, their model of collaboration and relationships with each other will also be taken into consideration. This would be crucial to establish a comprehensive perspective on the sociological development of the field. As community ethics plays an fundamental part in FLOSS digital art, it is therefore important to observe how it has been put into reality amongst them. It will also draw comparisons between groups with contrasting ideologies, highlighting the rich dynamics which propel the movement.

Lastly, the section will document various events relating to FLOSS digital arts. By doing so, it will illustrate the types of social outlets which exist for artists. Artistic festivals, workshops and academic conferences are some examples which can be seen regularly. They are set up to celebrate and raise the awareness of its practice. Through these events, artists are able to exhibit their works and more importantly, to convey their beliefs to the public spectators. As a result, the success of these events could also indicates the general acceptance of FLOSS digital arts.

As an active practitioner, the overall documentation will aim to offer insights into the current state of affairs in FLOSS digital art from a first person's perspective. In other words, factual information is gained through direct experiences accumulated from collaborating with others and being involved in the field. There are obvious advantages and disadvantages associated with this method of collecting research data. However, since this is a mixed mode research, the approach based on the personal practice would seem more appropriate than otherwise.

1.6 Practical component

Several criteria have been critical in finding a suitable project to be the practical component of this research. To begin with, the potential project should allow the theoretical element and the personal artistic practice to converge. In other words, it should display clear continuity and coherence between all elements within the research. Furthermore, it should reflect the research findings in a realistic context, and open doors to future investigations. Crucially, its result should be an original and useful contribution to the field. Lastly, it would provide an opportunity to further develop and establish the personal artistic practice.

To fulfill such requirements, the project must firstly involve FLOSS and identify a specific focus context. Moreover, it should aim to take on an active role in the field, so as to deepen the understanding of its internal development. The project, therefore, will be in a better position to make a visible and beneficial change to the artistic community. This will consequently expose it to other more experienced practitioners and as a result, their invaluable expertise and feedback will enable the personal practice to reach a higher level.

Since the beginning of this research, the artistic practice has developed an increasing emphasis on the use of a particular software program named Pure Data. Extensive experiences have been accumulated by using Pure Data to create customized performance environments and teaching it in workshops. As one of the most popular and versatile FLOSS creative software, Pure Data (Pd) has a large user community and has been deployed to create all types of art works. Pd is thus the natural candidate to form the basis and context of the practical project. The knowledge already gained in Pd would enable the project to be in-depth and highly focused, whilst the size of its user community would allow the project outcome to have real impact on others.

The project, therefore, will involve taking part in the current development of Pd by implementing new and useful improvements. By doing so, it will have a chance to be at the heart of an active community. Such an opportunity can lead to a closer examination of various sociological aspects of Pd. There are two

aspects to the originality of the project: the new features added to the existing program and the extended possibilities in using Pd. In other words, through the implementation of proposed improvements, artists will be able to incorporate Pd in ways that were previously impossible.

In short, the practical project will firstly review the design shortfalls in the current version of Pd, then identify the most suitable course in which changes can be made. It will clearly outline features to be implemented, as well as describe the rationales and potential impact behind the new improvements. Furthermore, the project will exceed its technical nature by looking into the internal social structure and historical developments in Pd's community, particularly in comparison to the general ideology of FLOSS. This would hopefully give an objective interpretation to any existing problematic areas and point out how they can be avoided in the future.

1.7 Objectives

Having introduced the relevant aspects and elements of this research, its objectives can now be succinctly defined in the following points.

- To analyze the current development of both digital arts and FLOSS, and examine the relationship between them.
- To document the ongoing activities in FLOSS digital arts in Western Europe.
- To provide an original contribution to the field of study by taking part in the development of Pure Data
- To further develop personal creative practice

The first three objectives consequently become the focal points in each of the following chapters. The investigations into contemporary digital arts and FLOSS will be the main concern in the subsequent chapter, whilst the third chapter provides a concrete documentation to illustrate the latest development in the

field. The final chapter will present the practical project in detail, and discuss various issues encountered.

1.8 Compositional example

A composition in the form of Pd patch will now be given as an example of the personal artistic practice in generative music. Furthermore, it will also briefly demonstrate the functioning of Pure Data in the context of sound synthesis and audio signal processing. This example will hopefully offer a good starting point for the rest of the research that follows. In this section, the following font conventions have been adopted to denote computer codes (in C), Pd objects and messages.

- `typewriter text` - Text in typewriter font means C code. Thus this is not C code, `but this is C code`.
- [`pdobject`] - typewriter text wrapped in square brackets represents a Pd object, where the name of the object and its possible arguments are inside of the brackets.
- < `pd's message` > - Objects in Pd works together by passing messages between each other, a message in Pd is denoted by typewriter text wrapped in angular brackets.
- [`pd name_of_a_subpatch`] - whenever a pd object is presented with its name beginning with "pd" followed by a space, this means the object contains a sub pd patch inside. As a result, a patch can contain many sub-patches and each sub-patch can have its own sub-patch.

Before discussing the technical content of this composition, it is crucial to state and examine the motivations behind this piece, in order to provide it with the appropriate overall context and justification. The purposes behind this composition can be broken down into the following points. First, experimentation in the use of simple generative techniques to produce an aesthetically interesting

automated sound composition. Second, to compose an abstract sound piece in which samples of recorded sounds are not allowed.

There is a wide range of commonly adopted generative compositional techniques in which the degree of complexity also varies dramatically. However, on closer examination, one soon realizes that many of the complex methods are often built upon a collection of smaller identifiable processes which function in co-operation to complete its complex design. More specifically, the use of randomization and probability has been implemented with high regularity as parts of large systems[22]. In fact, all computer operating systems have random number generators built in, and it plays an important part in the system's operation. The prime motivation and goal of this composition therefore is to use simple techniques - randomization and probability - exclusively and attempt to create a complex sonic composition. As a result, a better understanding of the nature of generative compositions can hopefully be obtained.

At the time of this writing, the personal compositional practice has been focused on the use of recorded sounds for some period. Therefore, the second motivation of this composition is to forbid the use of samples from the process of composition. The challenge of not using any pre-recorded sound will temporarily decouple the practice from its existing mode of composition and hopefully allow greater diversity to emerge. In other words, sounds used in this composition are strictly synthesized.

As previously mentioned, the core of this composition relies heavily on the ability to obtain random values in order to control or synthesize the necessary sounds. As the result, the following two mechanisms in Pd form the most basic building blocks of this composition.

- [`random`] object. The random object has two inlets and one outlets. The left most inlet triggers the object to output a random number via its left outlet while the right inlet sets the range which the returned random numbers are limited to. In addition, the range parameter can also be

[22]For example, Genetic Algorithms typically use randomization and probabilities for operations such as mutation and initialization

specified in the object's creation as the addition argument. For example, [random 100] will return any random values between 0 and 99.

- [noise~] object. The noise object generates white noise, which is sound that has equal energy across the entire audio frequency spectrum. The object outputs the white noise through its outlet. The inlet is not in use. In digital sound synthesis, white noise is achieved by outputting a randomized floating-point value between -1 and 1 at each sample. White noise can be implemented through the following C code:

```
static t_int *noise_perform(t_int *w) {
  t_float *out = (t_float *)(w[1]);
  int *vp = (int *)(w[2]);
 int n = (int)(w[3]);
 int val = *vp; // setting the seed value for random numbers

  while (n--) // dsp block loop
    {
      //scales the random number between -1 and 1
      *out++ = ((float)((val & 0x7fffffff) - 0x40000000)) * \
               (float)(1.0 / 0x40000000);
      // generates a new random number
      val = val * 435898247 + 382842987;
    }
     *vp = val; // store the random number seed
      return (w+4);
}
```

Having introduced the two fundamental building blocks of this composition, the Pd patch of 'Hypothetical_Waves' can be shown below:

Figure 1.1: Top level patch of Hypothetical_Waves

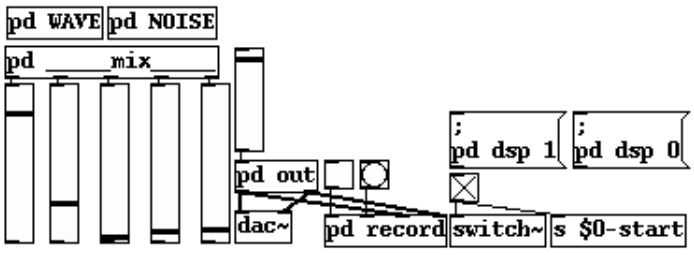


Fig.1.1 illustrates the general anatomy of the composition and brief descriptions of each part: [pd WAVE] and [pd NOISE] contains sound synthesis and processing sub-patches and [pd mix] contains mechanisms which are responsible for structuring the processed sound. In addition, [pd mix] has five vertical sliders connected from it, which indicate the current volume status of each of the

processed sound. [pd out] controls the overall volume of the patch and can be adjusted through the vertical slider connected to it. [pd record] enables the user to record the composition as sound files. <;pd dsp 1> and <;pd dsp 0> switches the global dsp engine on and off while the [switch~] object, controlled by a toggle button, enables or disables the local dsp functions within the patch. The following sections will further examine the vital components of this composition.

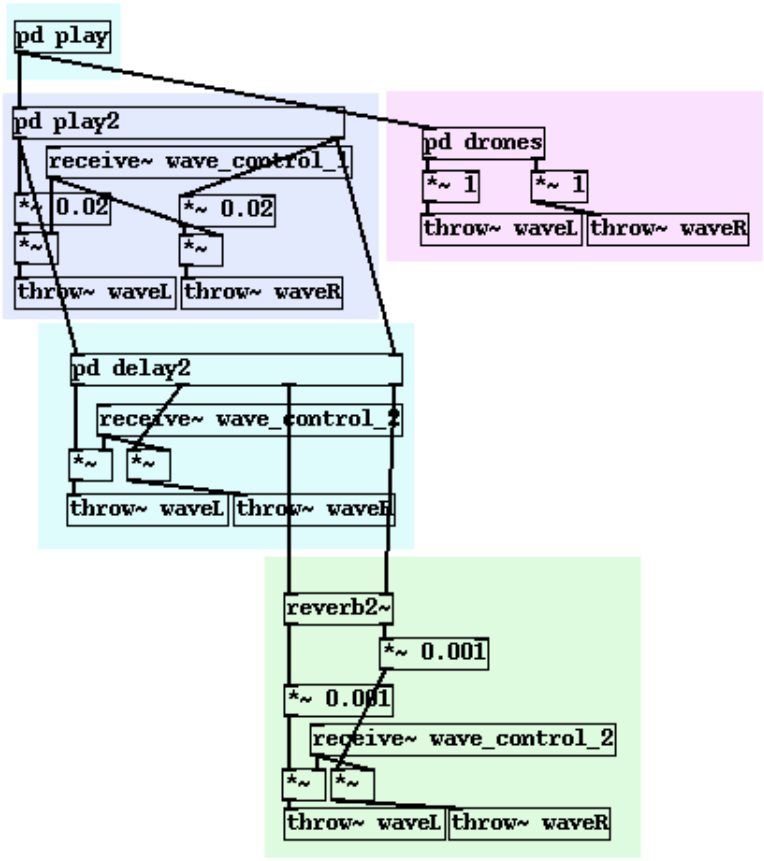
[pd WAVE]

[pd WAVE] is one of the two sub-patches responsible for the generation and the processing of the sound. Its overall anatomy can be illustrated in the snapshot taken at the top-level subpatch of the [pd WAVE].

In [pd WAVE], the chain of sound processing starts at the top where [pd play] generates the raw signal and each of the color blocks below accepts it as the input to be further processed. The input and output relationship between each color block can be understood by following the dark connecting cables which run between them. While each of the color blocks represents the individual sound-processing method and has its own sub-patch, it is worth taking a closer look at how the unprocessed signal is generated in [pd play].

[pd play] generates interpolated random audio signals with weighted probability over the amplitude value. To achieve this, three [random] objects (as highlighted area above) are used together in such a way that the first [random 10] connected to [moses 1] at the top produces a one in ten probability for the [random 200] objects at the left hand side to be triggered while the [random 200] at the right will be triggered for the rest of the time. Furthermore, unlike [random 200] portion at the left, [random 200] portion of objects at the right has a [* 0.2] at the end before connecting to the [pack f f]. As a result, this produces a series of random amplitude values that are most likely to be in the range of -0.2 and +0.2, but may be of any value between -1 and +1 with a one in ten probability. In order to interpolate between consecutive amplitude (y) values, the time (x) value also needs to be specified to indicate the number of

Figure 1.2: Top-level subpatch of [pd WAVE]



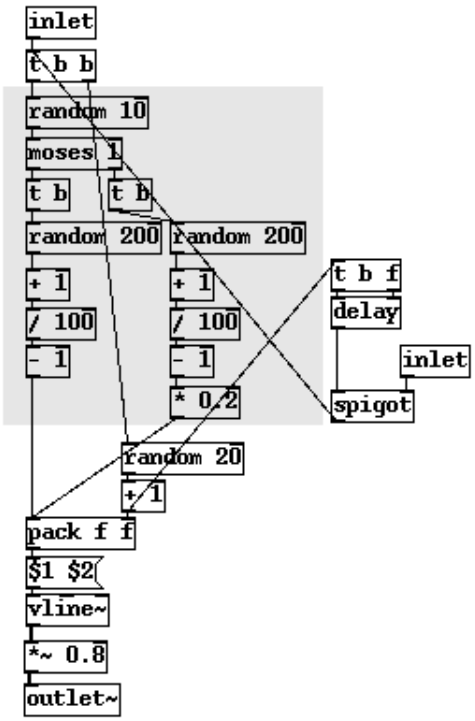
steps to reach the target amplitude. This is generated by [random 20] below the gray block. As a result, between any two amplitude values, it can take any time between 1 to 20 milliseconds to complete the interpolation. Both the x and y value are then given to the [vline~] object as input to produce the interpolated audio signal. Notice that this particular sub-patch is iterated indefinitely until the user switches the composition off at the top-level patch.

[pd NOISE]

[pd NOISE] is the second component responsible for the sound processing and its design can be illustrated below.

Following the same programming principle as in [pd WAVE], [pd NOISE] also has

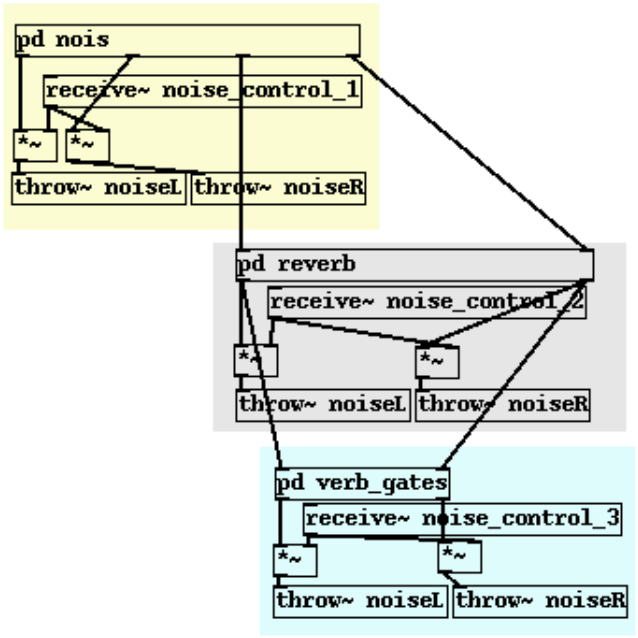
Figure 1.3: Interpolated random wave generator



a hierarchical sound processing chain with each color block signifying a distinct stage of signal manipulation. However, unlike [pd WAVE], [pd NOISE] makes use of the [noise~] mentioned previously as the sound source, instead of generating an interpolated audio signal. The white noise generator is implemented in the [pd nois] as figured in following.

Note the two identical strands of objects starting with [rand~] are connected to the outlet of [noise~]. The [rand~] object generates random values between -1 and +1 at the specified frequency and outputs the values as audio signals. The effect of these two strands of objects on the incoming signal (white noise) is identical to the noise gates commonly used in recording studios. In addition, the gated white noise is also mixed with a delayed copy of itself to give more character to the sound.

Figure 1.4: Top-level subpatch of [pd NOISE]



[pd mix]

Most of the time, each of the color blocks inside the [pd WAVE] and [pd NOISE] are programmed to output a distinctive sound (voice) and as a result, methods must be devised to mix and structure different strands of sounds to form the final composition. Two types of mixing mechanisms are therefore conceived; these are based on the use of random number generators (RNG), probabilities and saw tooth oscillators.

The design principles of the two mixing mechanisms are fundamentally the same but vary in the minor controlling method. Both mechanisms rely on the operations to turn a saw tooth wave (range 0 to 1) into a triangle wave (range also 0 to 1). The triangle wave is then used in turn to control the amplitude envelope of the sound. In this design, the higher the frequency of the saw tooth oscillator, the more rapid the change in amplitude will occur at the output. On the other hand, the lower the frequency is set to the oscillator, the longer it will take to notice the change in amplitude. The RNG is therefore implemented to generate the frequency values for the oscillator. Once a new

Figure 1.5: The use of white noise and audio gates

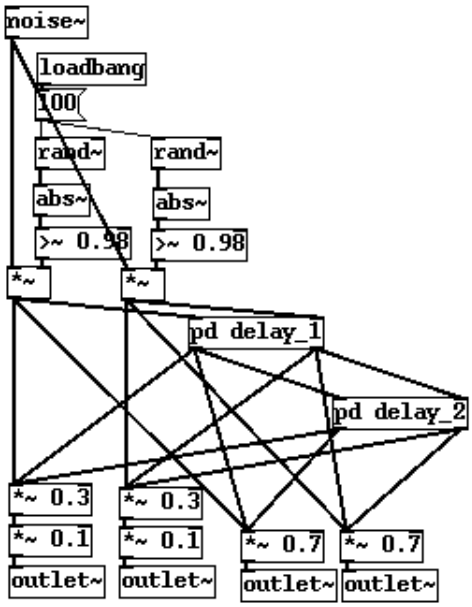


Figure 1.6: Mixing mechanism type one

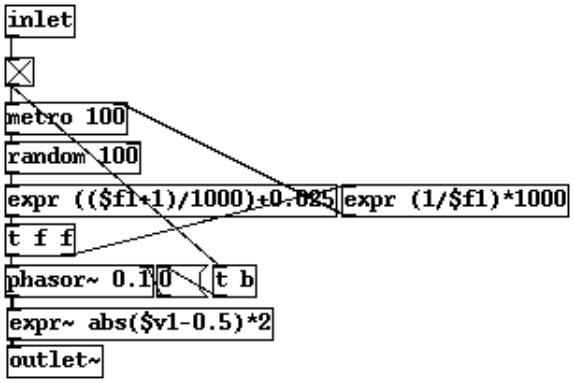
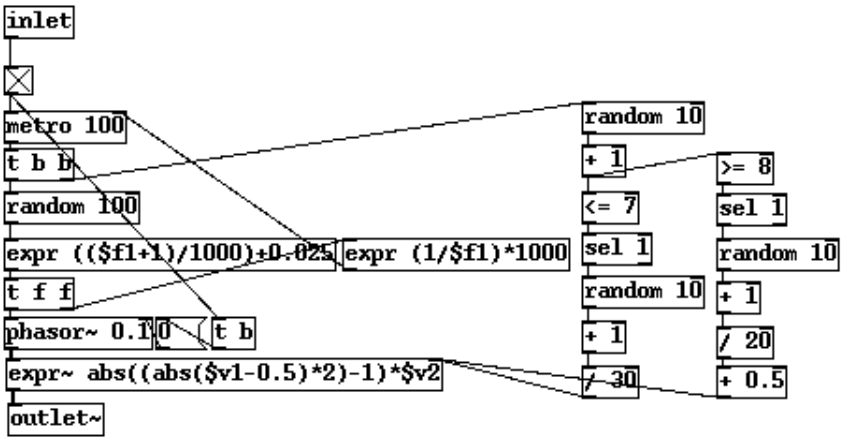


Figure 1.7: Mixing mechanism type two



frequency value is generated, the algorithm will also calculate the time it will take for the oscillator to complete one phase cycle (from 0 to 1) according to the new frequency. This timing information is used to regulate how fast or how slow it should [`random 100`] be triggered. Note that the range of random numbers in the mixing mechanisms is between 0.025 and 0.125, which represents the longest (4 minute) and shortest (8 second) possible total envelope time for each oscillation cycle.

The figure above shows the second type of mixing mechanism employed in the composition. Its design is mostly the same as the first type apart from the two modifications. First, in the operation of converting saw tooth waves into triangle waves, the resulted triangle waves are inverted. In other words, the triangle waves have a 25 present increase in their phase. As a result, the triangle wave will start at value 1 at the beginning of the oscillation cycle and reach 0 in the half way, and finish at 1 at the end of the cycle. The second change in design is the use of weighted probability to control the magnitude of the triangle wave. The probability implementation is the same as described in the [`pd play`]. As a result, the amplitude (0 - 1) of the triangle waves will most likely to be under 0.3 but will sometimes be over 0.5 at the ratio of eight to two.

This brings an end to the discussion on the programming aspect of this composition. It has demonstrated that simple generative techniques, with careful implementation, can be very expressive as compositional tools. In other words,

the effectiveness of a generative system is not measured by its complexity in design, but by how well it is implemented for the given task. As computers become increasingly capable and programing languages become easier to use, there is often a great temptation to design generative systems with great complexity. However, complex systems do not always produce good results and the effectiveness of very simple techniques should not be overlooked. In short, this composition made a conscious effort to avoid complexity in its implementation, and was aimed explicitly to explore the effect of simplicity and how it can be adopted.

Although this composition has achieved its aim of generating a variety of interesting sounds, it however falls short in producing a sense of progression and development in its structure over time. This quickly becomes apparent when listening to the patch over a longer period of time. Many programming strategies can be employed to overcome such a problem and should be investigated in the future.

Chapter 2

Digital arts and FLOSS

2.1 Digital Art

In modern societies, computer technology has infiltrated almost every aspect of human activities[1], and artistic practices are of no exception. The historical account and impact of new technologies - such as computers - on modern art practices has been a popular subject in various critical writings[2]. For many contemporary artists, regardless of their discipline, computers have become a vital apparatus in their creative process. For this reason, the ways computers are incorporated in the creative process are as varied and wide ranging as the artistic disciplines using them. To describe such kind of art practices and works, the term "digital art"[3] has become increasingly widespread and popular in recent

[1] In Nicholas Negroponte's 1995 publication entitled 'Being Digital'[25], he clearly outlined the extent and influence of computer technology over the functioning of modern society. Despite the book was first published over a decade ago, his views are still remarkably vivid and valid. For example, he suggested the widespread of accessible digitized information contents(music, video, photos, literature), the popularization of asynchronous communications(email, on-line messaging/chat), and the development of on-demand media services(YouTube, Last.fm). All of which can be observed in the current trend involving personal computers

[2] For example, Margot Lovejoy provided a lengthy discourse on technology and art in her book "digital currents"[22]

[3] Wikipedia on digital art: http://en.wikipedia.org/wiki/Digital_art

This thesis fully acknowledges the recent debates on the validity of referencing wikipedia

years.

The notion of incorporating computers in the creative practice is by no means innovative in today's artistic climate. For instance, looking up "digital art" on an Internet search engine such as Google quickly reveals the popularity of this terminology. Disregarding the precise context of how the term is used for the time being, the search results nevertheless indicate the general acceptance of such a method of artistic expression. Thus, it appears that there is a rich and active realm centered around "digital art" for artists and mainstream media. However, the widespread usage of the term has also led to a generalization of what it might imply. In other words, without further examination, the term can too often be vague or even misleading. This part of the thesis therefore aims to investigate the wide spectrum of creative practices in digital art.

2.1.1 Types of digital art

The use of computers in any creative discipline can, at present, be categorized in two ways. Firstly, computers are employed to simulate traditional creative apparatuses or environments. Secondly, computers are seen as instruments in their own right, exploring new possibilities through programming. Although the boundary between these two methods of practice are not always clear, they do, however, point to two common scenarios where computers are employed in artistic practices. In this thesis, the former mode of usage is referred to as "simulation based practice" while the latter as "programming based practice"

Simulation based practice

New technologies often first find their applications in improving the efficiency of traditional means of production. This notion, therefore can be seen as the foundation of simulation based practice. In other words, new technologies are

articles in the academic context. Therefore, the purposes of wikipedia references should be clarified at once. In this thesis, wikipedia articles are referenced only to either reflect the general perceptions on certain subject, or to provide factual information on subjects that typically evolve rapidly.

implemented to enhance the way artists work with the skills and knowledge they are already accustomed to. As computers possess the ability to emulate any procedures that can be logically expressed, this makes them particularly suited in this context. As a result, it has become very common to use computers in such a manner to assist creating works of art.

Because domain specific apparatuses are recreated in computers, through the use of various software and hardware which resemble a conventional working environment or tools, artists can therefore transfer their skills into the digital domain. The fundamentals of these artistic disciplines remain relatively unchanged, only the ways in which they are executed and expressed now have a different form. A clear continuity in both the mode of practice and aesthetics can usually be observed in this transition from the conventional to the realm of the virtual. For instance, through the simulation of a traditional sound recording studio, one can produce music in just the same way as in a real one. Digital photography and imaging is another good example.

Efficiency aside, flexibility and practicality are two other qualities highly rated in this type of practice. For example, many conventional tools can be programmed into a single software package which allows artists to explore the new use of these tools which would otherwise not be possible. Furthermore, following the previous analogy of a sound recording studio, almost every facility of a studio can be reproduced within a laptop computer. This gives far more practical means to musicians who have no access to the conventional music production environment.

Due to the nature of such an approach, the kind of creative disciplines adopting it generally already have their own established mode of practice and artistic concerns. Therefore, the center of this type of practice still remains at its traditional means and values. In other words, what is adapted or replaced by computers has a very direct reflection to its predecessors. This by no means denies innovations occurring in such contexts, however, computers are still largely used as means of production. Although the adaptation of new technology might have a fundamental impact on how a given practice is executed, the aesthetic

concerns and objectives still remain[4].

Artists adopting this approach can generally be identified as 'end users' of software tools or programs[5]. Furthermore, the relationship between artists and software tools is often simple and straightforward. This is because artists are typically not involved nor responsible for the creation of the tools they use. Such a categorization by no means undermines the extent of influence software can have on artists of this type, it simply points out the fact that artists did not have to develop the technology in order to use it in their practices. A simple analogy would be that a novelist is not required to be a mechanic to assemble or repair his or her typewriter.

Programming based practice

Their programmable nature is arguably the most significant aspect of computers, which distinguishes them from any other past man-made artifact. Because of this, any logical processes can be realized in the form of computer algorithms or programs. While some artists have been adopting computers to simulate traditional tools, others have aimed to create artistic works through the practice of computer programming. In other words, programming-based practice specifically explores new possibilities which may be achieved by using computer codes.

In any creative practice, artistic expressions are often restricted by the limitations of the tools used. In other words, artists are not able to produce works beyond the capacity of the medium they use. Since any conceivable idea can be materialized in the form of computer codes, programming thus offers a tremendous amount of freedom and a array of possibilities for artists. The liberation from various physical constrains of the conventional creative medium can be seen as the most attractive quality in creating works of art through computer

[4]For instance, a sound (or video/image) editing program might give new ways to manipulate and edit materials, the underlying goals of this tasks, such as best possible sound quality and compositional arrangement, clearly resemble traditional skills.

[5]Software such as the Adobe Creative Suite for digital image manipulation and Logic for sound recording

programming.

Programming, in this context, can be seen as a process in which artists translate their creative ideas into computer languages, resulting in a complete program, or collections of computer codes, which may be executed by computers. As computer codes are a formalized set of logical instructions, the process of programming often gives artists the ability to express their subjective concepts in an objective and rigorous manner. This ability to formalize - often abstract - artistic conceptions, can also be seen as an unique aspect of this mode of practice. Because of this, it can contribute to a methodological understanding, not only in the creative domain, but also gaining new insights into human creativity in general.

When computers and codes are adopted as creative media in their own right, they encourage new aesthetics and movements to emerge. The "Software Art"[6] and "Live Coding"[7] practices which emerged in recent years are good examples. The former, popularized in the late 1990s, shifts the artistic concern to focus on various aspects of software and codes. The concept of a given software, the structure of underlying codes and the social implications of a software are some common discourses in software art. Live Coding, on the other hand, considers the process of programming to be part of the performance practice. It demands that algorithms which are being programmed be shown to the audience, thus allowing them to gain insight into the performer's mental dexterity. Live Coding practice often demands great technical confidence and values the process of programming as equally important as the end results generated by algorithms.

The relationship between artists of this category and the software used is typically more complex. This is because artists also play the role of programmers or vice versa. An understanding of both traditional artistic disciplines and computer related knowledge is required to be innovative and productive. Therefore, it still remains less popular and mainstream than the simulation based practice. Furthermore, the demand for cross disciplinary knowledge has resulted in a rich mixture of artists that come from a wide spectrum of backgrounds ranging from

[6]Examples of software art can be found on http://www.runme.org/

[7]More information on Live Coding can be found on http://toplap.org

arts to science and engineering, to participate in the domain of digital art.

Comparison

Having identified two types of common practices in digital art, issues surrounding such categorization must also be discussed.

The categorization method presented above allows the term "digital art" to be further defined in a practical and straight forward manner. Moreover, this approach enables the meaning of "digital art" to be examined independently from traditional artistic disciplines. In other words, it can identify and compare the mode of practices in a generic way regardless of whether the outcome of the practice has the same "form"[8]. This generic approach in describing types of digital art by no means disregards their historical traces and roots that contributed to their development. It does however signify that digital art itself is a fully acknowledged artistic discipline in its own right, consisting of a wealth of activities whilst rapidly evolving.

Although the two strands of practices seems to have an opposite notion in adopting computers in the creative process, their relationship is deeply intertwined. For example, many of the popular software used in the former type of practice were often first developed by "hybrid" artists involved in the latter category[9]. Moreover, the software environment has been developed to enable artists to program in a very intuitive manner and requires a small learning curve. As a result, artistic practices can, and often do reside in both categories. In such a scenario, the main focus and intention of the artists can help to further reveal the dominant influence from the two modes of practice.

Finally, this approach draws a clear distinction between mainstream digital art practice and its experimental counterpart. The term "digital art" as used by the mass media, software/design companies and commercial artists often

[8]Common forms are such as digital imaging, digital audio, digital video/animation, installation. More comprehensive overview on the forms of digital art can be found in the second chapter of the book "Digital Art"[28]

[9]Early examples of such software are such as the "Digital Image Articulator" by Woody Vasulka, "Z-Grass" by Dan Sandin and "Easel" by John Dunn

points towards the first description. On the other hand, "digital art" in the independent or academic context often belongs to the latter usage of the term. This distinction is vital to the subsequent parts of this research, as it is largely derived from the programming based practice.

2.2 Current digital art practices

While the previous section outlined common modes of practice, this part of the chapter aims to reflect on the current climate in digital art by presenting and discussing some of the current trends and developments.

2.2.1 Generative art

In digital art, artists create generative procedures to produce aesthetically pleasing works. This can be achieved according to artists' novel conception, or through the means of recreating existing systems found in other domains. Generative systems have been applied to almost every types of art forms. This includes computer generated images[10], animations[11], structural forms[12], sounds[13] and even literature[14]. Lastly, live audio/visual performance is another domain in which generative art has been applied to[15].

As a result, generative art has gained increasing popularity in the domain of digital arts. Many artists, researchers, exhibitions and conferences[16] have all gathered under the name of generative art and as a result, it can be considered as one of the most mainstream and publicized genres of digital art. Despite the fact that it is widely recognized and has rapidly evolved, many aspects of generative

[10]William Latham, http://www.doc.gold.ac.uk/∼mas01whl/

[11]Karl Sims, http://web.genarts.com/karl/

[12]Celestino Soddu, http://www.celestinosoddu.com/

[13]John Biles, http://www.it.rit.edu/ jab/

[14]Scott Turner[39], http://www.pbm.com/oly/tag/srt.html

[15]Nick Collins addressed various issues surrounding this context in his paper 'Generative Music and Laptop Performance'[7]

[16]The International Generative Art conference in Milan is a prime example of such event. http://www.generativeart.com/

art still remain controversial and are yet to be defined. The following sections therefore aim to provide a closer investigation into the nature of generative art by examining its fundamental notions and characteristics. Then it will attempt to establish an overall taxonomy that defines the area of computer-based generative art. Such an analysis, will aim to not only reveal what the practice entails but also establish a methodological overview of the generative systems and the algorithmic processes involved.

Fundamental notions and characteristics

"Generative" means the ability to originate, produce or to evolve[17]. The word is often associated with the notion of growth and development. For something to be "generative" therefore must involves procedures that are executed for production and perhaps to progress. Natural evolution is a good example of a generative system, as it produces unique biological designs for the survival of species, thus achieving evolutionary divergence. However, as being "generative" does not usually imply the ability to anticipate and to foresee, there is often an inherited unpredictability in generative systems. The uncertainty in generative processes is also evidenced by natural evolution, as novel biological designs are not produced through planning but by natural selection and the 'survival of the fittest'. In short, there are three fundamental notions associated with the word "generative":

- The ability to produce
- Potentially a sense of growth over time
- The inherent unpredictability

The appearance of a plant as a result of its growth is also a good analogy of generative systems. A plant's seed contains nutrients and genetic materials for it to germinate. It then further develops into roots, branches and leaves, which make up the plant as a whole. Although this process of development is universal

[17] According to The American Heritage Dictionary of the English Language, Fourth Edition and Chambers Modern Dictionary, 1999

to all plants, no two plants have the same visual appearance even if they belong to the same species and grew under the same conditions. In other words, the growth process is generative in terms of the observable form of plants.

Following these notions on generative systems and algorithms, their general characteristics can be identified through the following three points, which will be further clarified and discussed subsequently.

- They must have the capacity to produce a diverse range of outputs which are not explicitly derived from the input.
- Their output (or their operation) may give the impression of continuous development.
- A degree of unpredictability and uncertainty lies in the functioning of algorithms, or the results they generate.

The first characteristic may be seen as the most fundamental quality a generative algorithm must possess. The key to this characteristic is the consideration of the diversity of a given algorithm's output. Because all computer algorithms "generate" output by manipulating the input, they can all be seen as "generative" without emphasizing diversity. Moreover, an algorithm must not solely rely on its input to achieve diversity of the output. In other words, the capability of producing a diverse output should be implicitly derived from the internal design of the algorithm.

The second characteristic often refers to the type of generative system whose operations are directed or goal-orientated. In other words, the behavior of the algorithms is not random but carefully guided by functions which cause them to work in certain ways. By using such guiding functions, generative systems are capable of producing observable improvements or developments over time. It is worth mentioning that the sense of continuous growth is usually the byproduct of the generative processes rather than the direct result. In the previous analogies of plants' development and natural evolution, the notion of growth is not explicitly specified in their abstract operation, but results from the incremental accumulative changes the systems are capable of producing. Many

popular programming techniques in generative art are good demonstrations of this characteristic. For instance, genetic algorithms produce accumulative results in order to deliver the optimized solution to a given problem. Furthermore, the learning period in neural networks also depends to a certain degree on the ideology of development.

The final characteristic stated may be further clarified by briefly discussing the motivations of applying generative systems in creating works of art. There are at least two common purposes: to resolve artists' creative blocks[18] and to create novel results, which will amuse or surprise both audience and creator. Both of these objectives require algorithms to produce outputs that are unanticipated. In other words, generative systems need to have the capacity to create unpredictable or even "creative" results. The unpredictability in virtual generative systems can be achieved in a number of ways such as incorporating random variables, adopting probability analysis or through simple deterministic rule-based systems. Evidence of the first two methods can be found in the typical design of genetic algorithms, where the occurrence of mutation is typically governed by random variables. Furthermore, the selection scheme in which chromosomes are chosen to reproduce, is often carried out by probability analysis on the fitness of individual chromosomes[19]. The third method relies on the vast number and high speed of repetitive executions on simple rules to give the illusion of being unpredictable. In other words, systems of this kind are essentially deterministic but operate on a scale or frequency that is beyond human comprehension or ability to forecast, and therefore appears to be unpredictable. Cellular Automata and The Game of Life are prime examples of this type of operation. A more detailed investigation of various generative techniques will be the focus of the subsequent section.

[18] In describing the project entitled Experiments in Musical Intelligence, David Cope pointed out *"I began Experiments in Musical Intelligence in 1981 as the result of a composer's block. My initial idea involved creating a computer program which would have a sense of my overall musical style and the ability to track the ideas of a current work such that at any given point I could request a next note, next measure, next ten measures, and so on. My hope was that this new music would not just be interesting but relevant to my style and to my current work."*[8]

[19] The implementation of these two methods in genetic algorithms result in the exploration of possible permutations and recombination on given elements and thus is able to create novel arrangements.

It is worth stating that none of the three characteristics listed above should be accounted for as the conclusive description of generative systems and processes. They are simply judging criteria that should be considered together, or in combination. Other ways can certainly be employed to define virtual generative systems. For example, Phillip Galanter proposed the use of complexity theory[20] to define and contextualize generative art[14]. Other approaches also include considering the degrees of control that users might have over the system. However, the characteristics identified in this section offer a universal and unbiased framework in which all possible generative systems are included.

Algorithmic examples

Having examined the fundamental notions and characteristics of generative art, common algorithmic techniques should now be presented, allowing a comprehensive and concrete understanding of its practice.

Randomization

The use of a pseudo random number generator (RNG) is no doubt the simplest technique in generative algorithm design. While fulfilling the first and the third characteristics stated above, randomization typically produces poor results in achieving the impression of "development". This downfall is rooted in the general design and objectives of RNGs, which is producing unpredictable, and thus disconnected outputs.

Randomization alone, is therefore not capable of generating aesthetically interesting results. This is mostly because the output is so disorderly that no perceivable pattern can be extracted. Therefore, the output can essentially be considered as 'noise'. Due to this, randomization is typically used in conjunction with other algorithmic techniques. For instance, in combination with

[20]Complexity theory studies the intricate relationships between individual components within a emergent system. Peter Coveney and Roger Highfield gave a illustrative account of this new science discipline in their book "Frontiers of Complexity"[9]

conditional or relational statements, one can achieve a greater balance between static patterns and noise.

Probabilistic

Probability is another common technique employed in generative system design. Through statistical analysis, this type of algorithm achieves unpredictability in the results it generates. Moreover, because the analysis is context dependent, the output thus typically displays a more coherent sense of continuity than that of randomization.

One of the most well-known techniques of this genre is perhaps the Markov chain algorithms[21]. In this technique, the probable occurrence of a given event against the event that precedes it, is calculated. A matrix, often referred to as the transitional matrix, is then used to store such probability of occurrence over all possible events in a given context. The transitional matrix can be generated according to the input of the algorithm in real-time, or from an existing database of events. In the realm of artistic application, the Markov chain is very useful in creating "variations" of data that all share a certain degree of resemblance. Furthermore, this technique can also be performed with varying degrees of complexity which govern such "resemblance" of data. This is achieved by including more than one preceding state in building the transitional matrix.

In short, the probability methods allow variations to be produced against a certain "theme", expressed through probabilistic values. The "theme" could be specified explicitly, or obtained from analysis such as the Markov chain. Although these types of algorithms are capable of generating both diverse and coherent results, they often lack perceivable development or progression over time.

[21] Markov chain is a well studied and documented technique in the context of computer music. see The Music Machine[34] by Curtis Roads for more detail

Nonlinear systems

Artists have long been fascinated by such types of systems because of their chaotic behavior. A common property of this type of system is the sensitivity to the initial condition or the input. This is commonly known as the 'butterfly effect'. The result generated can range from entirely predictable to seemingly non-deterministically random. Furthermore, the transition between the two states of output are hard to anticipate and control. For this reason, artists often adopt this type of system for its "expressive" nature and the wide spectrum of output it is capable of producing.

Although not directly related to formal chaotic functions, another common type of nonlinear technique found in digital art is the use of positive feedback. Feedback occurs when the output of a closed system is directed back to its input. A positive feedback means that the looped back output encourages the system to further escalate and take itself away from its equilibrium. Similar to the adaptation of chaotic functions, positive feedback is also liked by artists for its rapid and dynamic nature.

Although this type of system is often desirable for its nonlinear nature in producing diverse and unpredictable outputs, it is also notoriously hard to gain fine control over. Thus, results derived from such a method are often dramatic, but lack a well articulated structure and sense of progression.

Fractal

Fractal, in its broadest definition, describes a type of geometric shape which is extremely fragmented, with each subdivision approximately or exactly identical to its larger structure. This key notion in fractal is generally referred to as self-similarity. Thus any level of magnification consists of approximately the same degree of detail , and when compared, fractal is therefore often considered to be infinitely complex. The shape of the coastline is a classic example of fractal. Regardless of segment and scale from which the coastline is being observed, it will always display a similar structure and shape.

The 'self-similar' quality is also what often attracts artists to adopt such technique. The use of fractal allows different scales of structures sharing the same characteristics to be generated. This is highly desirable, as works of art often consist of layers of structure requiring a degree of coherence for them to be perceived as "meaningful". For example, the MusiNum[22] is a simple algorithm that produces identical sequences of numbers when its output is read sequentially or every second, forth, eighth number and so on. Such sequences can then be mapped into musical notes and thus produce not just coherent melodies but also harmonies.

It is worth noting that certain types of chaotic functions also relate to fractals for the self-similar properties they both possess. The strange attractor found in chaotic functions, for example, produces outputs that are fractal. Moreover, it has been discovered that some chaotic functions contain higher order self-similarity than that of fractal.

Modeling

Unlike previous examples, modeling based techniques are not entirely derived from abstract mathematical theories and constructs. Instead, their origins can be traced back to existing functional systems often found in the physical world. In other words, this type of approach aims to mimic the behavior of existing systems thus reproducing its capability in order to achieve certain design goals.

Evolutionary algorithms are a prime demonstration of this type of technique. Although they further contain several branches of developments[23] which are all different in detail, they all aim to emulate various mechanisms or effects found in natural evolution. The main interest of this approach thus often lies in achieving an adaptive system that can "solve" specified problems in a given context with minimal human intervention. For instance, the mechanisms of mutation are

[22] http://reglos.de/musinum/

[23] Subsets of evolutionary algorithms are such as genetic algorithms, evolutionary programming, evolution strategy, genetic programming. Peter Bentley and Corne provided a comprehensive review of the artistic applications of evolutionary programming in his 2002 book 'Creative Evolutionary Systems'[1]

frequently simulated to achieve greater dynamics in the output. The notion of competitive survival is also often modeled to create "intelligent" agents where their interactions display global emergent behavior.

Artificial neural networks are another good example. As the name suggests, they are based on replicating the functioning of neuron cells in the brain of living beings. Brains have the ability to accumulate past experiences and use them to make new adaptations in problem solving. This therefore becomes the main attraction of artificial neural networks. In other words, the aim is to create adaptive learning systems that can produce solutions responding to the change of conditions in the given context.

The adaptiveness and the goal-orientated heuristic behavior are often what attract artists to employ modeling as means of creation[24]. In other words, they are not only capable of producing a diverse range of outputs, but also display a clear notion of continuous development. Furthermore, the characteristics of this type of system can also give an impression of "purposefulness" when their operation is under observation. This is a significant departure from the aimless wander in the search space often associated with other types of systems.

This category of techniques are not only popular in their artistic applications, they have also been the subject of extensive scientific studies and interests. Research studies on this area often refer to it as 'Artificial Life.

Rule-based

The last approach of generative process relies on devising simple rules which are to be executed at a vast scale or rapid intervals. The scale of operation often plays a crucial part in the diversity and effectiveness of the end result. For this reason, the simplistic nature in its design would usually require the execution to exceed a certain measure in order to obtain a satisfactory result. In the artistic context, the scale of operation can be measured through following two criteria: first, the magnitude required for the system to reach a 'dynamic' state, and

[24]Early pioneers in this field are artists William Latham and Stephen Todd. They have publish their collaborative effort in the book "Evolutionary Art and computers"[37]

second, spectators' cognitive ability in predicting the behavior of the system or even the underlying rules which govern it.

Cellular automata, or the 'Game of Life'[25] in particular, is a good demonstration of the notion above. In the game of life, the state of a given cell is determined by its surrounding neighbors[26]. For the system to produce aesthetically interesting results, it would therefore require (A.) enough cells for their interactions to be effective and (B.) executed in the scale for the dynamics of the system to be observed efficiently.

Another well known algorithm of this type is called the Boids, or also known as the swarm behavior algorithm. In Boids, simple rules[27] that governing the motion of an object are first established. These rules are then uniformly applied to a collection of objects. As a result, a global effect that mimics the vivid movement of bird flocks can be achieved.

The Lindenmayer system, also known as L-system, is also commonly mentioned in the rule-based system. L-system specifies a set of rules[28] in which structures

[25]Originally devised by British mathematician John Horton Conway in 1970

[26]The rules for Game of Life are:

1. Any live cell with fewer than two live neighbors dies, as if by loneliness.
2. Any live cell with more than three live neighbors dies, as if by overcrowding.
3. Any live cell with two or three live neighbors lives, unchanged, to the next generation.
4. Any dead cell with exactly three live neighbors comes to life.

[27]Proposed by Craig W. Reynolds in his paper entitled "Flocks, Herds, and Schools: A Distributed Behavioral Model"[32]

1. Collision Avoidance: avoid collisions with nearby flockmates
2. Velocity Matching: attempt to match velocity with nearby flockmates
3. Centering: attempt to stay close to nearby flockmates

[28]A simple L-system rule can be described as below:

Variable used: A, B

Initial seed: A

Rules: A transforms into AB, B transforms into A

Results: A, AB, ABA, ABAAB, ABAABABA, ABAABABAABAAB ...

This simple rule produces the well known Fibonachi sequence.

can be developed. When applied with several iterations over an initial seed, complex shapes and forms can be generated. To this end, L-system is frequently used to produce structures that resemble the forms of trees and plants.

The underlying principle of a rule-based system can be summarized here: by applying simple local rules over a network of objects or iteratively over an initial condition, a diverse global behavior can emerge. In other words, the output of this system is greater than the sum of its parts.

Conclusion

Following the examination of generative art's fundamental concepts and the investigation of various common techniques, a more concrete and methodological perspective on its practice in the digital domain should now become apparent. However, some issues surrounding its practice need to be mentioned.

The practice of creating works of art through formal procedures and systems existed long before the term 'generative art' was coined and used by contemporary artists. As noted in the previous chapter, the idea of producing generative, ever changing music was already present in ancient civilization. Traces of such an approach in producing creative works can also be found throughout the history of artistic development across the world. In the context of western art music in particular, examples can be found as early as the 11th century by Guido d'Arezzo[29]. Mozart was also known to have devised a "musical game" named 'Müsikalisches Wurfelspiel', where piano minuets can be composed from a chart of small melodies and a series of dice throws. Modern composer Xenakis, is often considered to be one of the early pioneers in exploring the territory of music derived through the means of mathematical procedures. His book "Formalized Music"[41], profoundly influenced the application of the probability theory to the composition of stochastic music. More recently, various artists and composers from the minimalism movement have also had a very procedural approach to their works. Series of sculptures by Sol LeWitt and the musical

[29]By designating the five vowels to the notes of an ascending scale, d'Arezzo was able to create musical melodies from a given text.

compositions of Steve Reich[30] are particularly noticeable representations.

Referencing the two modes of practice previously identified, generative art can therefore can be seen as a "combined" practice, consisting of established aesthetics and new contextualization in the realm of computer programming. In other words, although generative art in the digital domain often has a strong emphasis on developing customized creative systems, its origins by far precede the popularization of digital art.

The significance of a given creative practice can often be measured by the unique expressions and values it brings forward. For example, the atonal and serialism movements in modern western classical music had a profound influence on the practice of musical composition. It can be argued that, in the context of generative art, its original artistic vocabulary and perspective are elusive and often vague. This is mostly because it shares much in common, both technically and artistically, with different practices in digital arts. As a result, its definition and relationship with others remains ill-defined.

This could be the most serious shortcoming in the current development of generative art. Its ambiguity could lead it to encompass other forms of practice, which might, therefore, result in a lack of differentiation in the broader context. On the other hand, its "inclusiveness" can lead to the misunderstanding of its original focus and concern, and ultimately being misused or misinterpreted.

It is no easy task predicting the next stage of evolution for generative art. However, if its popularity continues and it is eventually adopted by the mainstream practices, it would be extremely hard for it to take on a more consolidated position in order for a more concrete, clearly defined perspective and practice to emerge.

[30] One compositional procedure Reich utilizes is called 'Phase', whereby identical musical entities (rhythmic or melodic) and played against each other with a degree of offset in time. As a result, these entities will oscillate between different stages of phase, thus creating complex musical structure.[26]

2.2.2 Code as creative medium, Algorithm as instrument

The proliferation of computers in arts have also affected the necessary skills required to operate and program them. In the early years, programming a computer required a great degree of knowledge and effort. The languages[31] used to program computers were hard to understand and master. Progressively, modern computer languages began to be invented and adopted by more people. Modern languages such as C[32], were far less abstract, thus making computer programming a much less demanding task. As a result, programming was becoming increasingly accessible and widespread. Today, not only are modern computer languages still commonly used, many other new languages also exist and are specifically designed for real-time audio/visual manipulation in the creative context. As a result, programming a computer to manipulate sounds and images no longer requires years of training and a high degree of technical knowledge.

For digital art practitioners, computer programming has moved away from being a highly knowledge-specific and abstract task, to being a widely accepted and accessible activity. Artists write, exchange and share codes to further articulate their creative expression in the digital domain. Computer codes have thus become integral parts of their compositional practice.

The phenomenon of 'code as composition' has shaped contemporary digital art practices in many ways. This section will attempt to investigate those changes and examine their implication and effects. Based on the current relationships between practitioners, computers and codes, it will hopefully highlight future developments if such phenomena continue.

The investigation will begin with a global view to focus on changes in contemporary digital arts as a whole. It will then shift towards a more specific view based on the perspectives of individual practitioners, examining how coding has influenced their creative practice.

[31] For example, assembly language was used in early computer programming.

[32] C was invented by Dennis Ritchie in 1972 at the Bell Telephone Laboratories. It was originally designed to be a general purpose, procedural based programming language.

Before proceeding any further, it is worth mentioning some examples of programming languages that are now currently widely used by artists. This will hopefully further clarify the term 'computer codes' in the context of this thesis by presenting the following software.

- **Pure Data** Pure Data (Pd) is a real-time graphical programming environment for audio, video, and graphical processing. Originally developed by Miller Puckette and Co. at IRCAM. The core of Pd is written and maintained by Miller Puckette and includes the work of many developers, making the whole package very much a community effort. http://PureData.info

  Pd's graphical programming nature makes it far more visually intuitive to learn and program than the text based programming language. This visual advantage and its user-friendliness make Pd very popular amongst digital arts practitioners.

  Furthermore, after years of collective development, Pd has now evolved into a programming environment not only designed for sound manipulation, but also for generating animations, processing videos, hardware interactions and many other extended possibilities. Besides Pd's user-friendly interface, this versatile multimedia capability is also what makes it very popular in the digital arts community.

- **SuperCollider** *"SuperCollider is an environment and programming language for real time audio synthesis. You can write programs to generate or process sound in real time or non real time. SuperCollider can be controlled by MIDI, the mouse, graphics tablet and over a network via Open Sound Control."* http://www.audiosynth.com

  Unlike Pure Data, SuperCollider (SC) is a text-based programming language. In other words, instead of programming through the visualizations of interconnected objects on the computer screen, SuperCollider relies on a more traditional approach where lines of text are written by the user to define and manipulate sound. Because of this, SC is often described as being harder to learn and master. However, being a text-based language,

SC is more powerful in many ways and offers more flexibility in certain types of processing tasks.

SuperCollider is currently an audio only programming environment, although hardware interaction and network communication capabilities are available.

- **Other graphical based languages**
  - **Max/Msp** (http://www.cycling74.com/products/maxmsp.html)
  - **VVVV** (http://vvvv.meso.net/tiki-index.php)
  - **gAlan** (http://galan.sourceforge.net/)
  - **audiomulch** (http://www.audiomulch.com/info.htm)
- **Other text based languages**
  - **Common Music**
  - **Jsyn/JMSL** (http://softsynth.com/jsyn/)
  - **Csound** (http://www.csounds.com/)
  - **Processing**
  - any languages that have Audio/Synthesis API or libraries such as C, Perl, Python, Ruby, etc.

Changes in digital art

The technological nature of digital art often contributes to a wide spectrum of disciplines ranging from art to science converging in its practice. Although this multidisciplinary approach to computer-based art may be traced back to its early years of development, it is further helped by hardware and programming languages being readily accessible in recent years. This phenomenon can be clearly observed in the academic environment where many of the traditionally science-oriented institutions have begun to introduce creative elements in their training and art schools, opening up to technology based courses and research.

This increasing emphasis on the multidisciplinary approach in digital art has resulted in a diverse variety of collaborations and projects taking place in the field.

This not only helps to further perpetuate the practice itself, but also enables practitioners to gain new perspectives on all the encompassing disciplines.

Because of the unique combination of art and science, the multidisciplinary quality of digital art can have a more fundamental impact on the perception of knowledge by society. Art and science are often seen at opposite ends in the continuum of human knowledge. Art is inspired by creativity whereas science demands rigor, facts and experiments. However, with a closer look, such preconceptions are far away from the truth. Indeed, great artists are often rigorous in their techniques and attention to details, while ingenious scientists are highly creative in hypothesizing and proofing their discovery. With the increasing diversity of artists, scientists and engineers engaging in digital art, this could potentially dissolve this misconception. Such a unifying force is immensely valuable not only in the advance of knowledge but also by bringing a better perception of art and science to society.

Another major transformation in digital art is the change in the social context to which it belongs. In other words, digital art is now also flourishing outside of its conventionally associated institutional environment. Digital art is no longer an artistic practice limited to a narrow sociological scenario.

The direct impact of such an increase of social backgrounds is that digital art can now be seen, practised, exhibited and performed in a variety of locations ranging from academic conferences[33], artistic festivals[34], local enthusiasts meetings[35]to club nights[36]. Such an phenomenon prevents digital art from becoming inaccessible and ultimately sterile, by providing many different scenarios for its practitioners to showcase and evaluate their works.

Moreover, the increasing diversity of social outlets in digital art, may help the overall practice achieve a better theoretical and practical balance. In other words, theoretical developments may be evaluated and supported by practical experiences gained from other social contexts. Similarly, practical productions can be derived from cutting edge theoretical innovations.

[33]For example, International Computer Music Conference, http://www.computermusic.org
[34]For example, Ars Electronica, http://www.aec.at/en/prix/index.asp
[35]For example, Dorkbot meetings, http://dorkbot.org/
[36]http://www.perl.com/pub/a/2004/08/31/livecode.html

Lastly, the popularization of code as a creative medium also allows artists to engage with programming at different levels. For some artists, code itself has become as important as what it generated, and the process of programming is no longer hidden away from the spectator. Therefore, the aesthetic has largely, if not entirely, evolved around the structure, design, functioning and the intention of the code and the process of programming.

In short, computer codes have provided an invaluable common platform for practitioners of not only different professional disciplines, but also social backgrounds to co-operate. This change meant that digital art as a creative practice has grown out of its somewhat restricted early stage, into a more open and stable development. Furthermore, it also encourages the emergence of new kinds of aesthetics and practice to further explore the relationships between artists, codes and programming. Concrete examples of such a transition in the development of digital art will be given in the following chapter of this thesis.

Changes in individual practices

Code as an artistic medium has changed the way individual practitioners conceptualize and examine their creative works. For example, it has become increasingly common for digital art practitioners, who write and develop codes, to integrate different art forms into their creative works. In other words, the conventional boundary which defines and separates different types of art forms such as sonic and visual arts, are disintegrating through the use of codes. To further understand this, it is worth taking a more detailed look into the possible roles played by algorithms in the creative process.

Typically speaking, two types of algorithms are involved in the process of composition. First, algorithms that generate useful data which will be employed to govern the structure of the composition. Algorithms of this type are discipline independent- they merely produce abstract data that contain structures and patterns. Second, algorithms which are capable of mapping and translating data to produce the final outcome. Such types of algorithms are discipline-dependent because they need to be programmed to deal with domain specific

parameters such as frequency, pitch and amplitudes of sounds. In short, algorithms of the first type can be seen as "behavioral algorithms", which influence the global structure of the work, regardless of the art form and of the final outcome. On the other hand, the second type of algorithms can be described as "mapping algorithms", which accept the structures given by the "behavioral algorithm" and map them into a domain specific parameter.

Such concepts of "behavioral" and "mapping" algorithms could be further understood through the following analogy. In western classical music, to compose music in the sonata form involves at least two criteria. First, a sonata is required to follow a relatively strict structural form - introduction, exposition, development, recapitulation and coda. Within such a structure, a composer has to obey all the relevant music theories such as timing and harmony in order to materialize the sonata form into the final piece. It can be argued that the sonata form, by itself, is abstract and does not carry any specific musical information, it simply serves as a template, and can lead the audience to perceived a sense of drama, progression and tension. The traditional ABA structure is a similar example. In other words, it can be argued that structural strategies transcend their musical context and can be applied to any other time-based art forms. On the other hand, musical theories which govern the rhythm, melody and harmony are strictly domain-specific and are not easily transferable to other types of art forms. In short, "behavioral" algorithms are similar to the structural strategies discussed above, whilst "mapping" algorithms reflect domain-specific knowledge such as musical theories.

Traditionally, the notion of structural strategies and domain specific knowledge and skills are inseparable in the practice of art creation. This is because practitioners must start their artistic training by learning contextual skills, then combining them with structural rules so as to produce works of art. Moreover, contextual skills typically require extensive periods of time to practice and demand tremendous dedication to acquire. For example, the process of mastering a musical instrument or a type of painting technique is often a life-long affair.

Because artists are tightly bound to domain-specific skills, different types of art forms remain fairly mutually exclusive to one and another. Indeed, artists of

a given artistic discipline may not easily transfer their skills to other art forms due to their individual mode of practice.

However, codes as creative medium significantly increase artists' ability to conceptualize and express their work beyond such conventionally exclusive boundaries between various art disciplines. This is largely due to the fact that in the digital domain, regardless of the art form practiced, the ability to program is the one skill common to all practitioners and code itself became an universal medium to all. This has allowed artists to create works that incorporate other types of art forms previously not available to them due to the lack of appropriate "vocabulary". For example, one could create works containing digital sounds and images governed by the same behavioral algorithms. In short, programming as a mode of practice encourages a lateral thinking that sees all art forms as one in the digital domain, instead of isolated artistic disciplines. This can potentially lead artists to the discovery of "meta-art" which encapsulates all forms of art, thus revealing new understanding and the perception of art in general.

Furthermore, through codes, artists have become more aware of other types of artistic disciplines and acquired knowledge about subjects that they are typically not familiar with. Such phenomena has also taken place when electronics became readily available and musicians were eager to adopt and devise their own electronic instruments. In order to do so, musicians not only had to understand how traditional instruments were built, they also had to acquire the knowledge of electronics, to implement their design.

As a result, for artists wishing to work with codes, it is foreseeable that apart from their artistic development, especially in scientific and technology related skills may become increasingly important and dominant. In fact, Xenakis once pointed out *"the artist-conceiver will have to possess knowledge and resourcefulness in domains as varied as mathematics, logic, physics, chemistry, biology, genetics, human science, and history- in short a kind of universality, but a knowledge founded, guided oriented by and toward forms and architectures"*[33] In short, having the ability to learn efficiently as well as balancing between technical possibilities and their artistic intent would become the key criteria for

digital artists.

End note

This part of the thesis has provided a comprehensive overview on the current practice of digital art. Moreover, it has also outlined some ongoing trends and their surrounding issues. The development of digital art has certainly come a long way since its early days. As the world of computing technology continues to evolve, digital artists will no doubt embrace such advances in both hardware and software,whilst exploring new territories.

One example of such a territory can be found in the increasingly high bandwidth wireless network connectivity. Through wireless connection, artists have already begun experimenting with the ways this could be utilized in the creative context. Independent media streaming, network data visualization/sonification, collaborative real-time performances and distributed computing are just some of the themes which have received extensive interest from artists.

Moreover, the increasing memory capacity of storage devices also allow artists to create works consisting of greater details and complexity. The introduction of high-definition video hardware and format is a prime example. Other technological advances such as multi-core architecture and powerful hand-held or miniaturized devices, are also attracting many artists' keen interest.

2.3 Free and Open Source Software

The Free and Open-Source software(FOSS) movement has gained increasing awareness in the public eye and in the mass media in recent years. This is largely due to the fact that many FOSS projects have focused on achieving convenient packaging, distribution and installation systems, leading to the rapid expansion of its user community, to include the non technical users. Moreover, FOSS has evolved to support not only an extensive range of legacy hardware but also a range of cutting edge and highly specialized systems. This has allowed FOSS to be seen and used in a wide range of situations and applications. Finally,

because of both the quality and innovations in many of the FOSS projects, they have been deployed and used to provide critical services in institutions and enterprises on a large scale.

All of the above reasons combined, have contributed to FOSS becoming an increasingly dominant force in the current computing industry. The perception of FOSS has dramatically changed from originally an isolated movement in the early years to being embraced and highly recognized by several key figures of the industry today. Furthermore, FOSS is now also being also being supported at a governmental level by many nations across the globe.

The impact of such a revolutionary movement can also be observed in the field of digital art. Like its effects in other domains, the influence of FOSS transcends its technical nature. FOSS's ideology also profoundly affected various sociological aspects in the field. For instance, it has influenced artists to establish alternative methods to co-ordinate activities, exchange ideas and distribute works of art.

Before the relationship between digital art and FOSS can be further examined, it is necessary to investigate various aspects of the FOSS movement. The aim of this section is therefore to achieve a comprehensive understanding of FOSS. This part of the thesis will provide both a historical account of the movement and its ideology. It will also identify and discuss several of the current issues linked to the widespread use of FOSS.

2.3.1 Terminology

As Free software and Open-Source software are two distinct movements but have much in common, these two terms are often used interchangeably. However, this is misleading and can potentially lead to incorrect interpretations. For this reason, two terminologies have therefore emerged to allow them to be mutually addressed and referred to. They are: FOSS(Free and Open-Source Software) and FLOSS(Free/Libre and Open-Source Software). In this Thesis, the term FLOSS will be employed in the discussions on the subject. FLOSS was chosen because it is the least ambiguous term of the two, as "libre" denotes the meaning

of free as in freedom, not cost. The differences between free software and open-source software, however, will be presented and clarified in the sections to come.

2.3.2 Ideology

Like any influential movements, FLOSS is derived from and built on a set of principles which are set up and followed by its community. These ideologies can be identified as below:

- Freedom in software
- Mutual collaborations
- Resource sharing
- Voluntary and distributed efforts

Freedom in software may be seen as the central ideology FLOSS rests upon. It essentially sets the starting point for the subsequent elements of the movement to be derived from. The 'freedom' in FLOSS include the use, modification, distribution and redistribution of software and its source codes[37].

The demand for freedom symbolize the fundamental objection to any external factors limiting users' right in software. As a result, this ideology is often seen as the most radical and hard to understand part of FLOSS.

[37]The free software definition (http://www.gnu.org/philosophy/free-sw.html) provided by the Free Software Foundation are:

- The freedom to run the program, for any purpose.
- The freedom to study how the program works, and adapt it to your needs. Access to the source code is a precondition for this.
- The freedom to redistribute copies so you can help your neighbor.
- The freedom to improve the program, and release your improvements to the public, so that the whole community benefits. Access to the source code is a precondition for this.

FLOSS believes that its progress should be propelled by mutual collaborations, rather than direct competition. In other words, cooperation is the key notion in its development model. Such a view can arguably extend beyond the domain of software, and can be applied to other contexts of human activities, even to society in general.

FLOSS also considers that the resources of the community should be shared to allow a free flow of information and knowledge. In other words, duplication of works is generally discouraged, just as any factors which may contribute to it. This is because FLOSS recognizes that human resources and efforts are valuable commodities, which should be devoted to novel contributions, not replicating efforts or maintaining complex systems which limit others' freedom. As a result, individuals can focus freely on their original accomplishments, built on the works others have achieved previously. This has allowed FLOSS as a whole to evolve in a very efficient and vibrant manner.

FLOSS generally considers the software development process a voluntary affair. Software developers and users contribute to the improvement of a given program on a voluntary basis. The FLOSS model of software development is therefore not influenced by external factors such as commercial pressure. Precisely because of this, the production and maintenance of software can be driven solely by innovation and its usefulness to the community. Such an ideology has led to many outstanding FLOSS projects to emerge and flourish in the computing industry.

Furthermore, this ideology allows the effort of development and maintenance to be evenly distributed. This has brought several advantages to software development in FLOSS. First, survival of FLOSS projects does not solely rely on a few individuals. Secondly, the evolution of a software is flexible and agile. In other words, branches of a given software can easily be created to suit a particular purpose, and the divergence can be consolidated later on to unify all the changes.

2.3.3 Historical accounts

The Hacker culture

The exchange and sharing of software and their source codes can be traced back to the hacker culture among university computer laboratories during the 1960s. At this time, personal computers were not yet invented, and the use of computers was based on users sharing the time and resources of a centralized computer, also known as the 'main frame' computer. Because of this, a sub culture began to emerge, which derived from lab member's fascination with computer programming and spending long periods of time writing software and exchanging codes, to demonstrate and further improve their skills. The word 'hackers', by its original definition, therefore refers to people who gain enjoyment and take pride in creating better programs or improving software written by others. One of the most well known location of the hacker culture at this time was the artificial intelligence lab at MIT, led by the renowned computer scientist and artist Marvin Minsky.

The main fascination of early hackers was not entirely derived from the notion of programming but more from the construction and understanding of formal systems. Programming simply provided the means by which hackers could build and manipulate such logical constructs. It was a common practice for hackers to immerse themselves in technical manuals to gain insight into the functioning of a particular hardware/software system in order to improve its efficiency.

Because hackers were strongly motivated to acquire technical information, the culture as a whole developed to have a very liberal approach towards the distribution of knowledge. As a result, it was generally discouraged to restrict access to resources and information. In many cases, hackers felt "obliged" to unlock information -with necessary means- so that others could access it.

It was extremely unfortunate that the mainstream media later misinterpreted the term "Hacker" by associating with it the increasingly common criminal offenses in cyberspace[38], which were unrelated to the original hacker culture.

[38]These criminal acts typically involves the illegal intrusions of computer systems over the

For this reason, the term "hacker" now often has negative connotations in the public eye, whereas the liberal attitude towards knowledge and the motivation in achieving technical brilliance failed to be recognized by the media.

One of the reasons which contributed to the original hacker culture having a strong sense of community and such a liberal philosophy, is the "gift culture" embedded in its practice. To gain the acceptance of peers or to increase one's reputation, one would have to make contributions to the community, either by creating useful programs or by solving an existing problem. In other words, through the exchange of 'gifts', one's skill is verified by the peers whilst respect within the community may be accomplished. This 'gift economy'[39] system was dominant throughout the hacker culture and can still be observed in today's FLOSS movement.

Author Steven Levy in his 1984 book[21] provided a detailed historical account on the original hacker culture and its surrounding technological and sociological issues. Levy had formalized -and possibly first used- the term, "Hacker Ethic"[40] as a set of principles which hackers value and follow. These principles described by Levy still find their context in the current FLOSS movement and are recognized by its practitioners.

The original hacker culture gradually ended during the 1980s. It was largely

Internet. Media such as Newsweek and CBS new started using the term "Hackers" in reference to these cyber criminals in the early 1980s.

[39]Although the gift economy model has commonly been used in explaining the economic foundations of FLOSS, its shortfalls and alternative approaches have been suggested by authors such as Steven Weber[19].

[40]Levy[21] identified the hacker ethic as the following:

- Access to computers - and anything which might teach you something about the way the world works - should be unlimited and total. Always yield to the Hands-on Imperative!
- All information should be free.
- Mistrust authority - promote decentralization.
- Hackers should be judged by their hacking, not bogus criteria such as degrees, age, race or position.
- You can create art and beauty on a computer.
- Computers can change your life for the better.

due to the fact that commercial companies, based on releasing proprietary software, began to emerge and become successful. Hackers hired by such companies were prohibited from exchanging information under non-disclosure agreements. Market pressure created by the increasing popularization of personal computers further restricted the sharing of source codes, and by this point, software had become a commercial commodity. Furthermore, since computer programs were now products companies relied on for financial gain, it was also forbidden to copy and redistribute them without authorization. In other words, both the programs and their source codes changed from being tools and knowledge belonging to the public domain, to commodities corporations traded for profit.

GNU and Free Software Foundation

In the effort to preserve the original hacker culture after it began to diminish, Richard Stallman initiated the GNU[41] project and consequently the Free Software Foundation(FSF). The goal of GNU is to produce a complete operating system in which both the software and its source codes are free from commercial restrictions. In other words, GNU gives back to users the freedom to study, modify and share software. The free software foundation, on the other hand, was originally aimed at facilitating the development, promotion and legal support for the GNU project.

The main motivation behind both GNU and FSF is to protect the freedom in software which were destroyed by the commercialization and the growth of the computing industry. In other words, the establishment of FSF officially marked the beginning of the free software movement, and the consequent development of FLOSS. As GNU and FSF were aimed to safeguard and advocate the practice of sharing software and codes, it can be agreed that FLOSS is deeply rooted to the original hacker culture that once flourished inside of the computer laboratories in university campuses. Because of this, Richard Stallman is often considered as the last true hacker of the era.

In order to obtain a complete operating system, the GNU project consists of

[41] GNU stands for GNU's Not Unix

a wide range of components such as software libraries, system utilities, compilers, kernel and other programs. Amongst these GNU projects, a few of them are particularly notable and played a crucial role in the development of GNU and the FLOSS movement. Firstly, the GNU Compiler Collection(gcc) enables programmers to compile their source codes into the final software. Second, the GNU C Library(glibc) provides common functionalities which software can access. Another prominent component of the GNU project is the GNU Emacs text editor. Emacs is also one of the first software written for GNU. It is an extremely versatile text editor which may be used for many tasks ranging from programming, composing documents, reading/writing mails to browsing the Internet.

The decision to produce emacs, gcc and glibc at the early stage of the GNU project proved to be highly profound and strategic on Richard Stallman's part. It meant that programmers were able to produce software based on the tools offered by GNU. This ability is critical because the resulting software are not influenced by any commercial licensing methods. In other words, it enabled the development of GNU to be completely independent from commercial software tools and packages. As a result, the freedom in software were preserved.

The Free Software Foundation, on the other hand, provided an invaluable support -not only to the GNU project but also to the free software movement. Founded in October 1985 as a non-profit organization, FSF had formalized the freedom in software in which it aimed to protect, and legitimize these protections in the form of the GNU General Public License (GPL). A countless number of free software programs were subsequently released using GPL, thus setting the standard in FLOSS.

Apart from the continuous work on the GPL in adapting it to the changes in the software industry, other FSF activities also includes publications, on-line project hosting, campaigning, and education to raise awareness of the free software movement in the public domain. Moreover, FSF has also become established outside of the USA, in Europe, India and Latin America. After over twenty years of operation, FSF continues to be a dominant force in FLOSS and still makes significant contributions to the free software movement. Many of the

GNU software projects still remain vital components to the FLOSS community, being highly used and regarded.

Linux

After several years in the development of the GNU project, one critical component that was still missing in Stallman's quest for a free operating system was a functional kernel[42]. Although much efforts had been made on GNU's part to produce such a kernel[43], its development was slowed down by various licensing and technical issues. In 1991, A Finnish graduate student in computer science at the university of Helsinki named Linus Torvalds released a unix-like kernel as free software under the GPL.

Linux was derived from a hobby project which Linus created to enable him to connect with the university terminal remotely from his Intel 386 personal computer. Linus quickly realized that several of the components[44] in this hobby project resembled many key features required by a kernel. He therefore switched the focus of the project to a different direction.

Perhaps the single most important decision Linus had made in the early stage of Linux is allowing anyone to have access to its source code via the Internet and consequently contribute to its development. Because of this, it was not only worked on by Linus but also by several other people who had found the project interesting. The developer and user community grew fast as Linux became increasingly stable and useful. Another significant decision taken by Linus was the compatibility between Linux and GNU software[45]. This meant that Linux would immediately gain its functionality when combined with software tools from GNU. Because of this, Linux has adopted the GNU General Public License since the release of its version 0.11 in December 1991.

[42]Kernel refers to a critical component within a operating system, its main purpose is to assign system resource, such as cpu, memory and I/O, between all programs

[43]GNU Hurd, http://www.gnu.org/software/hurd/hurd.html

[44]For example, a task switcher, file-system, etc.

[45]More precisely, the common compatibility between Linux and GNU is due to the fact that both are Unix compatible.

The combination of GNU software collection and Linux kernel finally yielded a completely free operating system. Linux filled in the last missing piece of the GNU project. This combination had become so popular that the term "Linux" is often used to refer to the entire GNU/Linux operating system instead of just the kernel itself.

Besides its technical contribution to the free operating system, Linux also demonstrated a different model of software development and social collaboration that is both highly productive and controversial at the same time. Unlike its GNU counterpart, Linux's development process is typically more frantic and apparently chaotic at times. Although both GNU and Linux rely on open community in software development, GNU's methodology is often described as being more goal-oriented and rigorously designed. Such contrast in the model of development was clearly highlighted by author and software developer Eric S. Raymond in his well known essay entitled "The Cathedral and the Bazaar", which he later published as a book under the same title[31]. Although the validity of Raymond's arguments in the essay are debated, notably by writer such as Nikolai Bezroukov[46], they nevertheless pointed out how the method of collaboration had evolved from that of GNU by Richard Stallman to Linux with Linus Torvalds.

Despite the debate around the methodology between FSF and Linux, there is no doubt that the contribution of both FSF and Linux transcends the realm of software design and distribution. In other words, they offered a different approach in which collaborations could take place, which embraces both freedom and the community. In fact, it can be argued that the sociological outcome and effects demonstrated by GNU and Linux far exceeded their original context, and such methods of collaboration and distribution started being applied to other fields of social activities.

[46]Bezroukov had addressed his response to Raymond's view in papers such as "A Second Look at the Cathedral and the Bazaar"[3] and "Open Source Software Development as a Special Type of Academic Research (Critique of Vulgar Raymondism)."[2]

Open Source Initiative

In 1998, several influential members[47] in the free software community began an initiative in which the term "free software" was replaced by "open source"[48]. Consequently, the Open Source Initiative(OSI) was formed in the same year to facilitate and encourage the transition. In the same way as the role of FSF in the free software movement, OSI maintains the official open source definition and provides certification systems for licenses[49] meeting its requirement.

The main reason behind such change is that the free software community had started to receive increasing attention from corporations with potential commercial interests. Open source advocates felt that in order for the free software community to progress into a new era and benefit from it, a less ambiguous and neutral term was needed, as neither interpretations of the word free was suitable in the corporate context. Free as in no costs, would give the false impression of being not profitable, whereas free as in 'freedom' carries strong social and political undertones.

The changes from free software to open source proved to be a success in the commercial world. Key figures[50] in the existing computing industry began to adopt to the open source movement, and several companies[51] were formed within the open source community that later became influential and hugely profitable[52]. This also resulted to some significant changes in the original free software community. The original free software movement therefore being no longer marginal-

[47]The key figures includes Todd Anderson, Larry Augustin, John Hall, Sam Ockman, Christine Peterson and Eric S. Raymond

[48]The original call to the community urging the change of terms can be seen in the following URL: http://www.catb.org/ esr/open-source.html

[49]St. Laurent offered an comprehensive introduction to various licenses within FLOSS in his book entitled 'Understanding open source and free software licensing'[20]

[50]for example, Netscape had released the source code of its web browser "Navigator" using open source. Companies such as IBM, Dell and Hewlett-Packard had becomes increasing supportive of open source in the recent years

[51]Companies such as RedHat and Novell are derived from within the free software community

[52]Technology journalist and author Glyn provided an compelling documentation on the relationship between GNU, Linux and Open Source movement during this critical period in his book 'Rebel Code'[24]

ized and isolated, new relationships were formed between the community and industry. Moreover, the commercial success dramatically raised the awareness of the initiative in the mainstream media and general public in a very short period. The movement therefore benefited and was further developed from this recent popularization.

In many respects, the initiation of the open source movement is based more on a strategic rationale than on the continuation of certain ethical beliefs derived from the FSF. In other words, the aim of open source is to take advantage of the current external interests and develop it in its favor. Although the strategy had clearly paid off, some side effects can also be observed due to the nature of the movement.

One of the side effects of this transition is that the adoption of open source has often been based on mostly economical and practical reasons for individuals and enterprises. In other words, the freedom in software that both the hacker culture and the free software movement once cherished had become less valued and recognized. Moreover, the free software movement has at times even been misunderstood and sidelined. This is especially noticeable in the context of the mass media where open source is presented without mentioning or acknowledging of the free software movement that preceded.

2.3.4 Free software and Open Source

The ideologies and activities of the original free software movement and the open source initiative have much in common. However, some subtle differences do set them apart. Despite disagreements between the two movements being minor, the implications have nevertheless been significant. This is because their changes occurred at a very fundamental level.

The main emphasis in the free software movement were the ethical issues surrounding users' freedom in software. FSF addresses the injustice in the use of software in society and attempts to remedy such inequalities by means of developing the GNU project and promoting the use of GPL. In other words, the free software movement had a strong social and political concern behind its technical

achievements. The goal of the free software movement was clearly identified to reclaim and promote freedom in software for users.

The open source initiative, on the other hand, had a much more practical starting point. OSI believed that the proliferation of open source software should be based on its technical merits and its development model. OSI's practical approach also remained self evident in its strategic departure from the free software movement. Because of this, 'freedom in software' in the context of open source is used as a means to obtain the best possible software, rather than the goal itself.

The interpretation of freedom caused the two movements to develop their own values, which separated them distinctively. This difference in values was addressed succinctly by Stallman in his essay[53]. He pointed out, *"For the Open Source movement, non-free software is a suboptimal solution. For the Free Software movement, non-free software is a social problem and free software is the solution"*. Through software, the free software movement address higher ethical issues in relation to intellectual property. However, it is often criticized for its progressive and at times naive approach. On the other hand, although the open source movement has been successful in recent years, its pragmatic methods has sometimes resulted to corporations taking advantages of the open, collaborative community.

While both FSF and OSI continue to be actively involved and cooperate against proprietary software, they are now generally referred to by using the term FOSS(Free and Open Source software) or FLOSS(Free/Libre and Open Source software). This not only acknowledges both their achievements but also allows a generic term where they can be mutually included. With the increasing popularization of FLOSS, it is important to not misunderstand the free software and open source as one - they are two separate movements, and the distinctions between them are clear.

[53]Free Software Free Society[36], page 55

2.3.5 Current FLOSS climate

The success of FLOSS has brought changes to the movement. Moreover, its influence is extending further outside the domain of software development and distribution. One such change is the shift of concerns from developers' perspectives to that of the users' experience. Two factors contributing to this transition can be identified. First, because of the increasing awareness of FLOSS among the general public and mainstream media, more non-technical users have been drawn into adopting FLOSS. In other words, FLOSS's user community has expanded significantly to include users of all abilities, and has now reached a critical balance between software developers and normal users. Second, since FLOSS had already proved its technical advantages over the past years, a change of development to focus on users experience would be required to reach a wider audience. Because of this, FLOSS developers on the one hand need to start adjusting to the changes in the nature of its user base and on the other hand, providing solutions enabling FLOSS to become more accessible to the general public.

The most noticeable example of this shift is perhaps the release of a GNU/Linux distribution named "Ubuntu" in October 2004. Ubuntu's main goal is to make the use of GNU/Linux operating system as straightforward as possible. In other words, one can install, maintain and perform various tasks without substantial technical skills. Although such attempts have been made by other modern GNU/Linux distributions, Ubuntu is by far considered as the most successful effort to date. Another example of a FLOSS project that is hugely endorsed by the general public is the Mozilla Firefox web browser. Since the release of version 1.0, FireFox has now become the second most widely-used web browser. It is estimated that it has more than seventy million users world wide.

Other recent changes in FLOSS also include the development and support for hand-held devices such as mobile phones, media players and gaming consoles. Fueled by the advances in miniaturized computing technology and wireless connectivity, personal electronics have become highly demanded in the market place. Many manufacturers[54] therefore started to adopt FLOSS as the op-

[54]One of the most well known mobile phone manufacturer, Nokia, has produce a entirely

erating system for such products. In some cases, devices were developed with strong emphasis on the FLOSS integration. For instance, companies openly embrace individuals wishing to write software and perform customizations for the product[55]. In other words, manufacturers are now attempting to adopt not just the software but also the open methodology of FLOSS into various aspects of their products.

This change has highlighted the proliferation of FLOSS from its conventional usage in servers and desktop computers to the next generation of smart personal devices. This would enable FLOSS to be deployed in a wider range of contexts whilst allowing all supported platforms to communicate. As a result, users can not only benefit from a unified software environment but can also encourage innovative uses of FLOSS.

Another noteworthy example involving FLOSS is the ongoing OLPC[56] (One Laptop Per Child) project. Led by MIT media lab and several key figures[57] in the technology industry, the aim is to produce fully functional laptops at very low cost. They will then be distributed for free to children worldwide, especially focusing on those in developing countries. OLPC therefore hopes to minimize the technology divide in the future in the world by giving children the opportunity to access knowledge and communicating with one and another. Features of these laptops includes high-resolution color screen, audio input/output, wireless connectivity and extremely low power consumption. GNU/Linux was naturally selected as the operating system for the device, so to achieve long term sustainability, as well as low production cost.

The effect of FLOSS can also be observed outside of the software and computing context. As mentioned previously, the underlying collaborative methodology of FLOSS has brought forward a successful alternative to social activities in general. Its gift economy based culture which values distributed community, re-

Linux driven hand-held Internet Tablet in 2005. This device (Nokia 770 Internet Tablet), has gained much popularity since then and a second version was release in early 2007

[55]A South Korea company, Hampers Holdings, produced a Linux based hand-held gaming console named GP2X. The company actively encourages users contribute to the devleopment of game and software for the console.

[56]http://laptop.org/

[57]Google, News Corp, AMD, Red Hat, Brightstar and Nortel

source sharing and voluntary efforts, provides an efficient backdrop where great accomplishments may be made. This is a stark contrast to the conventional model of cooperation based on profit and rivalry.

One of the most notable influences of this kind may be found in the phenomena of the on-line knowledge database called "Wikipedia"[58]. As the name implies, it is a combination of the word "wiki"[59] and "encyclopedia". Its aim is to create a wealth of knowledge through voluntary editing and peer review. The database of information created can then be freely accessed and referenced. To date, wikipedia contains millions of information entries and exists in many different languages. Following the success of Wikipedia, many similar projects followed, each serving a different and specialized content.

Social activities such as wikipedia also highlight another important aspect of distributed collaboration. Because of its open structure, participants generally contribute to the work in progress according to their ability and expertise. As a result, a high quality and standard of the overall work can be achieved without centralized management and task assignment.

Apart from the model of collaboration, the FLOSS's approach to licensing has also been exported outside the realm of software. The establishment of the Creative Commons(CC)[60] in 2001 is the most recognizable attempt of this movement. Through a series of customizable licenses, author/artists could release any creative content enabling others to modify and redistribute it freely. Instead of the "All rights reserved" limitations imposed by the conventional use of copyright laws, creative commons allows "Some rights reserved" and encourages the fair use of any materials published under CC. Since the introduction of CC, more and more artists and institutions are adopting this type of licensing methods. This can be seen through the amount of creative material already accumulated and freely available on the Internet[61].

[58] http://wikipedia.org/

[59] Wiki is a type website that allows the visitors to easily edit and change available content on the page

[60] http://creativecommons.org

[61] For example, archive.org hosts a wealth of creative content licensed under CC thus make them freely available

With the success of projects such as wikipedia and Creative Commons, this approach of releasing and managing information is now generally referred to as "open content", following the analogy of "open source". Apart from the examples given above, the promotion of open content can also be observed in the context of academic publishing[62] and scientific data base[63]. Moreover, independent journalism is another sector which has been heavily affected by this phenomena, also known as Indymedia.

In short, FLOSS's influence on decentralized, distributed collaboration methodology may be observed in increasing aspects of social activities, some of them having yielded highly positive results.This approach has also had a huge implication in the context of contemporary digital art.

2.4 FLOSS Digital Art

A growing number of artists have been adopting FLOSS and an intimate relationship between their work and FLOSS can often be noticed. In other words, the merging of FLOSS transcends its practical implications and affects certain fundamental aspects of artists' practice. The general raising of awareness of FLOSS can obviously account for such a phenomenon. However, other reasons have also partly acted as catalysts in this alliance between artists and FLOSS.

The adaptation of FLOSS can be seen as a natural progression due to its successful development model and ideologies. FLOSS meets the demands of digital artists in providing innovative software tools, which are thus compatible in the practical context. Moreover, many of the principles in FLOSS closely reflect ideologies in art practices. This is most noticeable in the shared emphasis on the notion of collaborative works in a decentralized and open environment. As a result, the bond between artists and FLOSS also extends to the philosophical context.

[62] Formally known as Open Access (OA), http://en.wikipedia.org/wiki/Open_access

[63] One example of Open Access scientific database is called GenBank, in which a large database of genetic sequences is maintained and made publicly available. http://en.wikipedia.org/wiki/GenBank

The similarity between digital art and FLOSS can also be found in each of their current status. Digital art has grown out of its academic environment and has moved towards becoming a mainstream artistic discipline. At the same time, FLOSS has evolved from being a sub-culture - also originating from academia - to being publicly recognized and endorsed. In other words, they are both gaining popularity and undergoing critical changes brought by such transition. More and more artists are taking up the practice of digital art, exploring and defining the boundaries of art in the virtual domain, whilst the general public has become evermore familiar and informed with works of art expressed in the digital form. Having reached critical mass, FLOSS's popularity and implication is expanding exponentially, and the FLOSS community needs to address the surrounding issues to navigate this turning point.

This section therefore aims to examine the connections between digital artists and FLOSS. It will hopefully point out the contributing factors unifying them. The investigation will take into consideration the practical and ideological context, to observe the complementing elements and the similarities between them. It will also attempt to provide evidence to prove that their alliance has more advantages than with proprietary software.

2.4.1 Model of analysis

As software tools become an indispensable part in the practice of contemporary digital artists, an intimate dependency can therefore be found between artists and software. Software could significantly influence one's creative practice in almost every aspect. For instance, the capabilities of software have a direct impact on the limitations of what can be achieved by artists. Obvious technical constrains aside, software also affects other more subtle and implicit aspects of the practice.

In order to examine various aspects of FLOSS in the digital art context, the relationship between artists and the software they use needs to be understood methodologically. This thesis proposes a method of analysis following the development of software and how they are employed by artists. This model of

investigation therefore consists of the components below:

- The development
- The production
- The outcome

The development simply means the process of software being created. The word 'software' in the context of this analysis can have two meanings. First, it can refer to a program that is ready to be used by artists. Secondly, development tools such as programming languages and libraries are also considered as software. Regardless of which of the two reflect a particular practice, the key notion is to clearly highlight the part of software production and maintenance that artists are not responsible for - be it a complete program or the design of some computer language.

The production symbolizes the activities and interactions between artists and software. Different levels of interaction lead artists to have different roles. Artists could either be users of a given software, where works of art are based on the artistic use of such programs, or they could also be authors of a creative software. In a such scenario, the artful execution and design of codes become modes of creation.

The outcome is the end result produced by the combination of the two elements above. It recognizes the differences in forms that the end result may have and hence takes form. For instance, the outcome could be a static visual rendering or a recording of artists' conception. It could also appear as a software package expressing the artists' creative design. In the current context, the main interest is to identify what is produced by artists from their creative processes.

One important factor highlighted by this method of analysis is the relationships between artists and software in different stages. From the initial creation to the final result, it follows a top to bottom direction. Artistic practices, in this model, are therefore represented by the link between production and outcome. This also highlights the hierarchical nature in this model. As a result, any

changes occurring at the top would have significant effects on the consequent layers below. However, this does not necessarily mean that artists would be aware of the impact and influences caused by the software development. In other words, although not strictly part of the creative practice, the relationship between the development and the production layer is able to fundamentally influence the way artists work.

2.4.2 Current software development method

Having introduced the model of analysis, it is necessary to evaluate the conventional method of software production and usage according to the above criteria. Currently, most software is distributed using a copyright license and marketed as a commercial product. In other words, to use any copyrighted software programs, artists are required to purchase a license. Once the software is authenticated, the program cannot be re-distributed freely. In addition, the software is only sold in the binary executable form, without its source codes. This is because software companies safeguard the source files and use them to gain commercial leverage. Despite it being the most common licensing method, many issues and shortfalls can be clearly observed.

First, the interaction between the development and the production layer is rigid and inflexible due to the copyright agreement and the commerce driven nature of the software market place. This means that the ways in which software can be legitimately used and obtained is strictly limited by the licenses purchased. In other words, it does not encourage any other software outlets - which could further extend its user base - to take place. Moreover, companies can potentially exploit the popularity of their software and develop this into a market monopoly. In this case the cost of license fees is likely to increase, whilst the customer will have less influence and fewer options regarding the technology.

Secondly, as software programs are traded as packaged products, and users are essentially customers of the software company, this allows the development layer to dominate its relationship with the production layer. From the users' perspective, such a relationship creates a problematic dependency. In other words, users

have almost no control over the implementation and the policies of the software. For instance, users have no other choice but to accept the functionality of the software and have limited influence on its future developments[64]. Furthermore, there is no room for software customization and for adapting it to specific needs.

Thirdly, users only have access to the functionality of the software provided by the software company. This is especially troublesome in the context of artistic practice precisely because of its creative nature. Having a software tool with capabilities fixed and predefined by a third party, significantly constrains the possibilities in the practice itself. In other words, artists are limiting their practice with the features of the software. Finally, artists very often have to adapt their creative process according to the changes in the software used. For artists, this introduces unnecessary external factors in the development and shaping of their creative practices.

Finally, by using commercial software, the derived artistic works indirectly rely on software companies for their survival over time. If a software company is no longer operational or stops supporting a particular software package, any derived works are most likely to be lost in time. With commercial software, users are bound to the existence and the support of the software company for their works to continue being usable.

These issues have demonstrated how the development layer can have a critical and implicit influence on artists' practices. They have also outlined serious problematic areas in the current method of software production adopted by the majority. The usage of software is therefore biased and strongly in favor of companies producing them for commercial purposes, leaving artists with limited or little creative freedom. It is therefore necessary to research into alternative solutions.

[64] Although many software company pre-release BETA versions of a given software to selected user groups for feedback and testing, such a method is often limited in terms of the number of users able to take part. In addition, the exercise is frequently time-consuming.

2.4.3 FLOSS as alternative

Before the investigation proceeds any further, one can outline some criteria that a better software development model should fulfill:

- The new model should encourage flexible use of software with regards to its distribution and re-distribution.
- The new model should allow artists to influence the future design and development of the software.
- The new model should enable artists to customize the software tools according to their practice.
- The new model should ensure the longevity of the works produced with the software.

Fortunately, what the new model suggests is already present as the FLOSS movement. As stated previously, apart from permitting free distribution of both the program and its source files, the development model is governed on a voluntary basis. Because of this, FLOSS allows a new kind of relationship to emerge between software developers and artists. The differences quickly become apparent using the same analysis model above.

Because FLOSS has removed the conventional copyright restriction, the relationship between the development and the production layer has become much more flexible and fluid. For instance, the usage of software tools is only determined by their availability, rather than by licenses. This means artists can simply obtain programs from the Internet and use them as they wish. As a result, software tools proliferate quickly in the artists' community and programs are no longer traded as commercial products. Moreover, as the development layer in FLOSS consists of independent programmers collaborating via global networks, this approach is generally more liberal and democratic. This differs from the bureaucratic and hierarchical atmosphere typically found in commercial software production environments.

The elimination of the 'for-profit' commercial element in software development has had a more profound effect. The absence of software company means that developers and artists are able to communicate directly without interference. This allows for programs to be produced more rapidly and truthfully reflect the users' demands. The success of a given software is therefore measured by the size and activities of its user community, rather than by the commercial outcome. In order to increase the user base, developers have to make software reflecting the real needs of users, instead of implementations that are potentially more profitable.

In other words, FLOSS restores the inequality of the conventional user-developer relationship. Moreover, users may later become members of the developers' community. Such a notion is highly attractive from both users' and developers' perspectives. As users become developers, they are able to further influence the future changes in the given software. And as the developers community expands, tasks are more efficiently shared, thus achieving higher productivity and quality of codes, which in turn attracts more users. The user-developer attraction in FLOSS is an extremely powerful concept. This can generally be recognized as a positive feedback process with increasing returns, in which the accumulation of one factor helps the growth of another, thus further benefiting itself. In this context, the feedback process takes place between users, developers and the software produced. This makes the development process an open ended system guided by both users and developers. Although commercial software companies also have the means of complying with user feedback, the process, however is typically less transparent and flexible.

One of the effects of such a mutual collaboration between developers and users lies in producing a stable and robust software with collective debugging. Traditionally, the debugging process is governed by employed programmers with limited attention and concern for the specific software. In FLOSS, every user can potentially take part in debugging and testing the software. Consequently, the larger the user base, the more human resources can potentially be devoted in making the software stable. Thus, by removing the commercial elements and introducing mutual communication between users and developers, FLOSS

allows innovative, stable and flexible software to emerge.

By releasing the software specification and source codes in the public domain, the survival of FLOSS-derived work is not dependent on the original maintainer of the software. In other words, if the original author of the software stops supporting the development, others can still maintain the software. Moreover, other programs can also adapt to the open specification of FLOSS software, which would provide additional compatibility to further guaranty the longevity the works produced.

The advantages of FLOSS fulfill both the practical and conceptual demands in the context of digital art. As a result, FLOSS has become an increasingly attractive solution for many practitioners.

2.4.4 Practical example

The following section will provide an example to demonstrate the contrasting effects on the artists community between the closed, proprietary software and FLOSS. This example will use two widely employed software programs called Max/Msp and Pure data. They are very closely related to each other in their functionalities and therefore generally have the same target users, thus making them particularly suited for such comparison. Despite being proprietary software, Max/Msp has adopted the strategy of encouraging sharing resources and exchanging information within its user base. As a result, many artists highly value this community effect of Max/Msp and an increasing number of creative proprietary software now also follow this methodology. This comparison would hopefully point out that although Max/Msp mimics various characteristics of FLOSS, it would nevertheless still inherit the underlying issues of proprietary software.

Because Pure data is a FLOSS project, its development is not centralized and dictated by a few. This enables any capable artists/programmers to work on the current release of Pd and contribute to its improvements. As a result, various aspects of Pd such as hardware support and external libraries far exceed its counterpart. More importantly, artists are able to modify Pd from its source

codes to suit their needs, or even create different branches specializing in certain functionalities. The development possibilities of Max/Msp, on the other hand, are strictly limited by the company producing it. Although Max/Msp allows its application programming interface[65] to be publicly available, artists and programmers can at most only build external libraries, but not modify any core components.

From the users' perspective, both Pd and Max/Msp may at a first glance give similar impressions: both programs have an active user community and regularly exchange information in the form of mailing lists or on-line forums. However, this is also where the similarity ends between these two software. Although the emphasis on community generally has a positive effect on Max/Msp's users, it can also be argued that artists are led into a false sense of creative freedom. In other words, users are only allowed and encouraged to collaborate within certain limits[66]. This implication of user community is hugely different from that of Pure data. This demonstrates that despite the effort in introducing the values of FLOSS into proprietary software, the intrinsic problematics of closed source projects still persist, and merely become more ambiguous for users.

Despite the limitation of Max/Msp, its proprietary nature has had some advantages - most noticeably in the forms of software packaging, documentation and the sophistication of the graphical user interface. However, as Pd reaches a wider range of potential users, more developers are available to bridge such gap.

2.4.5 Conclusion

In reality, digital artists often feel the need to use several different types of software to fulfil various creative tasks. The practical advantages of FLOSS thus quickly becomes apparent. The innovative nature of FLOSS means that

[65]API is a source code interface that a computer system or program library provides in order to support requests for services to be made of it by a computer program.

[66]For instance, users are encouraged to develop Max/Msp external objects by using its public closed source API. Because of this, users are not able to customise their code beyond the capabilities defined by the API, as well as the inner workings of the program

artists are more likely to find available programs meeting their specific demand, or alternatively they could easily create customized tools. Also, because of the common practice of open standard and compatibility in FLOSS, artists may utilize such quality to have a collection of software tools which all work and communicate with one and another to enhance the overall functionality. As an example, all programs running on the GNU/Linux operating system follow certain conventions in design. For this reason, they can all be used in combination through a common scripting language[67]. In other words, common standard and compatibility contributes to FLOSS tools being extensively customizable. The more configurable software tools are, the more possibilities can be explored by practitioners to perfect their work.

Another crucial advantage of FLOSS arguably lies in its hardware support. For instance, a multimedia installation work typically involves elements such as computers, electronics, sensors and sound/light emitting instruments. As a result, coordinating these devices demands highly extensive and capable tools to control all the elements. It is not uncommon for FLOSS tools to run on almost any computer platforms. They cannot only be used on the latest PC hardware but also on more 'exotic' platforms such as recycled computers, hand held devices, game consoles and embedded systems. As a result, practitioners are able to carry out their designs on almost any devices within their reach.

In short, it is the flexibility and accessibility that sets FLOSS apart from commercial software. Although proprietary software tools offering such an extent of flexibility may also be found, they tend to involve high costs and require very specific types of hardware and software setups that may be harder to obtain. For this reason, FLOSS evidently exceeds its counterparts as a practical solution for artists.

Conceptually, freedom of expression is central to any of the creative practices, and digital arts is no exception. It is most crucial that the medium and the tools that artists employ allow them to fully express their artistic concerns. Although any creative apparatuses always have physical and practical limitations, no restraints should be imposed on their usage. In other words, artistic appara-

[67] Generally known as the bash script.

tuses should be used freely within their design limits. In the digital art context, computers are the physical instruments which have design limitations such as processing speed, physical dimensions and memory size. Although artists cannot create works exceeding such physical constraints, they should however be able to use the instrument in any way they wish within the limitation of the hardware.

With commercial software, such freedom of expression is illusive and intangible. Commercial software may have extensive features which are seen as having great capabilities. However, the limit can clearly be observed after one has exhaustively learned or used the software. In other words, artists are bound to the given features of commercial software to produce their work. FLOSS, on the other hand, has no such restrictions on the usage of software. Artists are free to explore the possibilities of FLOSS software and are able to redistribute any derived works. Moreover, freedom of expression embedded in the FLOSS model also encourages artists to experiment and innovate in order to further extend their practice.

In conclusion, FLOSS is highly compatible with digital art practices. FLOSS provides a stable and sustainable platform on which artists can build their works. In addition, FLOSS also enables artists to fully explore the possibilities in their practice by giving freedom to their creative needs. As a result, many FLOSS tools have been highly successful and widely used in the artists' community. Pure Data, is one of such example. If the success of FLOSS continues in the digital arts realm, it is foreseeable that it will become a mainstream solution in the future.

Chapter 3

Activities

3.1 Introduction

While the previous chapter provided a detailed investigation into digital art and FLOSS, the aim of this section is to document some of the current activities in the research context. It therefore has two purposes: to provide factual, ongoing examples supporting the discourse of the preceding chapter, and to produce an accurate record of the current affairs related to this research. The former will hopefully bring greater coherence and clarity to the overall thesis, whilst the latter will contribute to the subject areas with detailed documentation that future researchers may use as a source of reference.

This part of the thesis will begin by presenting various artistic collaborative groups, followed by several individual projects and finally present events such as conferences, festivals and exhibitions. This should encompass a wide range of practices and social scenarios in FLOSS digital art, for it to be representative. All documented activities were selected according to the criteria set in the previous chapter. In other words, they must coincide in the context of digital art with a strong focus on the ideology and adaptation of free and open source software. Most importantly, they must be currently active and evolving.

Being a mixed mode research with an emphasis on the practical aspect of the

personal creative practice, this chapter therefore primarily relies on first hand experiences as the main source of knowledge. It acknowledges some possible limiting factors in the information gathered. For instance, it has a restricted geographical context within UK and continental Europe. This is the result of both the physical location and the scope of the artistic practice this research is derived from. However, the practice's active involvement in the context of FLOSS digital art in Europe would nevertheless allow the documentation to reflect to the ongoing development in the field.

3.2 Collaborative groups

As both the FLOSS movement and artists value the effects of mutual and decentralized community, collaborative groups seem to be a natural habitat for digital artists who choose to adopt the FLOSS ideology. They work together, either physically or virtually, and share information and resources with each other. In other words, not only is technical knowledge being exchanged, so are artistic concerns and ideas. Moreover, they are also able to receive constructive criticisms within the collective. A typical trait of this type of artistic community is a rich dynamic on social and professional levels. For this precise reason, such a collaborative environment is usually captivating and thought-provoking, and is arguably an ideal backdrop for creative practice.

In recent years, these kinds of collective groups have been spawned all over Europe. They are typically independently formed, mostly without formal support from institutions or related authorities. As a result, they generally have a more "grass roots" and spontaneous characteristic than other forms of social outlets in digital arts. Some of these groups have distinct geographical bases, whilst others exist in cyberspace. This section will present a selection examples of some FLOSS digital art collectives in Europe.

3.2.1 Goto10

Goto10[1] is a non-profit organization founded in 2003 by Aymeric Mansoux and Thomas Vriet in Poitiers, France. The original goal was to produce regular and local alternative electronic music events. These performances invited both regional and international artists to showcase their works and engage with the local community. As young university students make up a large part of the local population in Poitiers, these events gradually generated much interest and thus proved to be popular. Goto10 quickly realized the potential of extending these performance programs, with workshops for artists to demonstrate their practices, and more importantly, to create hands-on learning opportunities for members of the public interested in digital art. These workshops also became successful, and were organized alongside the regular performance events. In this respect, Goto10 has successfully met the cultural demand for such kinds of activities, and thus created its own local community centered around contemporary digital arts. However, at this time, FLOSS was not an exclusive part of Goto10's operation and philosophy. This was evident by the content of their workshops, which occasionally included proprietary software such as Max/Msp. Goto10 has evolved significantly from the very beginning, these series of events are still being held regularly, and now also include an annual festival devoted to FLOSS and digital art.

Goto10 began to broaden its horizon in 2004 to feature a network of international artists as its core members. This effort symbolized Goto10's keen interest in taking on an active role in the artistic context. It therefore began to produce projects devoted to the development of various aspects of digital arts. Within this group of artists, strong affection for, and interest in the free software movement was the common characteristic shared between them. For this reason, FLOSS became the central principle in Goto10's ideology. In other words, not only are artists devoted to the FLOSS movement, the organization in general also has adopted an eager voice to advocate the use of free software in the context of technology related art practices. Since 2004, Goto10 began to establish itself in the field of FLOSS digital art in Europe. Goto10 and its members have

[1] http://goto10.org

actively collaborated, as a collective, with other similar groups and has now become recognized amongst its peers.

Having the two distinct parts - curatorial and artistic - co-exist and complement each other, was one of the main intentions in extending Goto10's activities On one hand, Goto10 engages with the local community by organizing regular events, which makes FLOSS and digital art ever more accessible. On the other hand, Goto10 experiments with technology via the projects it supports, which provide an essential platform for artists to innovate and create. The combination of these two elements creates a dynamic balance between reaching new audiences and developing creative technology. Moreover, it also makes the works of Goto10 respond realistically to the needs of the community it serves. For instance, software projects may be taught and tested in its workshops, with any feedback then possibly being used as an evaluation to guide their future developments, thus also preventing them from being inaccessible to the artists.

For the artistic part of Goto10, being able to present their work through the collective is an important element. In other words, artists are able to join forces and distribute their works in a unified manner. This is crucial in many aspects: It enables artists to have a more focused outlet for their activities, thus achieving a greater impact. This can significantly increase the sustainability of each individual artists' creative practice. For this reason, Goto10 artists generally work under the identity of the collective, rather than as separated individuals. This ideology also contributes to building strong ties between artists, thereby creating an intimate and trustworthy atmosphere amongst them. This community effect can be clearly observed between them, as they consider the works achieved as a collective to be greater than the sum of its parts. Furthermore, through Goto10, artists are able to express their ideas and conceptions more effectively and succinctly.

The artistic section of Goto10 currently consists of thirteen members. In contrast to its curatorial part, the artistic practices of Goto10 do not have a physical location: these artists reside in different parts of Europe, Canada and Asia. Collaborations and communications are achieved both via Internet channels such as a mailing-list and IRC. Artists also maintain an on-line repository system to

host various elements of their projects and source files, thus vital information can be effectively exchanged and become available for others. Goto10 also aims to assist other FLOSS artists - who are not members - by allowing them to have access to its resources. Internet based services such as web hosting, repository and streaming are a just few examples of what Goto10 offers.

Goto10 recruits its artists through the process of informal recommendation by existing members. Following the majority's opinion, a new position can be created democratically. In addition, there is an internal mailing-list through which these artists can communicate. Although Goto10 artists have active collaborations with other practitioners in the field, certain aspects of its operation, such as organization and development, remains confidential to its members. In other words, it draws clear policies in defining relationships with others. This approach allows the internal planning and discussions to be more focused and effective, without any unnecessary interference.

Having a geographically distant group of artists has also helped Goto10 to extend its reach outside of south western France. As its members often also work within their local artistic community, wherever they may be, this frequently results to Goto10 related events being at a local level. Thus the original objective to engage with local communities is further realized through virtual collaborations between Goto10 artists. As a result, small local communities which closely relate to Goto10, have started to emerge in cities such as London, Glasgow and Cologne. The blend of physical and virtual communities has proved to be hugely beneficial to the development of Goto10. The adoption of such an approach is perhaps one of the most significant decisions that Goto10 made from the very start.

Another aspect that distinguishes Goto10 from its counterparts lies in its approach to organization and in achieving goals. In many respects, Goto10 has clearly defined objectives, which have effects on both its internal structure and external perception. Although relationships between artists are often casual, the principles which they value are generally strict and precise. As a result, the collective has developed a somewhat uncompromising quality in its works and in its association with others.

In short, Goto10 is a growing collective built on many contrasting characteristics. This can not only be seen in its activities, but also in terms of its policies and practices. These qualities prevent it from being sterile and produce the dynamics required to propel the organization even further.

3.2.2 OpenLab

OpenLab[2] was founded by artists Dave Griffiths and Chun Lee in London, in late 2004. The frustration of having no physical social opportunities for London based FLOSS artists was the initial motivation behind the creation of the OpenLab collective. Despite the effectiveness of communications over the Internet, they both felt that the experience of being able to meet and interact in real life should not be missed. For this reason, the main goal of OpenLab is to provide an informal social meeting place for FLOSS inclined artists to converge and collaborate.

After the initial announcement of OpenLab, local artists responded with a high level of interest, and informal meetings thus started taking place. A series of meetings and discussions eventually led to the production of the collective's first public event consisting of an evening of live performances the following April. Since then, more events, including workshops, open demonstrations and exhibitions, have been organized by these artists. The size of the collective also increased as more artists in the capital learned about OpenLab through these events.

One of the fundamental philosophies of OpenLab, as its name suggests, is that all aspects of the collective are open and accessible by anyone. In other words, OpenLab attempts to apply the same ideologies of FLOSS into its functioning and organization. Instead of software development, these ideologies are now being applied to coordinate participating artistic practices, so that they may collaborate effectively. For instance, OpenLab's web page is a wiki, which means that anyone is able to edit its content. Furthermore, it also has a public mailing list where artists are free to register and post messages. All this

[2]http://openlab.pawfal.org

aims to facilitate and encourage dialogues between artists and thus mutually influences the progression of the collective. This open policy, has to a large degree, contributed to the informal and relaxed atmosphere between its artists and the character of the group as a whole. This characteristic of OpenLab has significantly influenced many aspects of its operation and development.

Following this ideology, any artists wishing to be involved with OpenLab can simply subscribe to its mailing-list , or contribute to its wiki page, to participate in the discussions and activities. In other words, OpenLab imposes almost no restrictions to its membership. The only criteria is the awareness and interest in FLOSS and its surrounding issues. For this reason, OpenLab has attracted a wide range of artists from different cultural and social backgrounds. University lecturers, graphic designers, programmers and students are just some of the professions found amongst the members of OpenLab. This policy constitutes a vibrant dynamic in its community and often brings contrasting perspectives to the group.

Another important ideology of OpenLab is its geographical emphasis. It believes that certain aspects of social experience are hard, if not impossible, to convey or achieve through virtual communications. For example, being able to practically demonstrate works and present ideas in front of peers allows the interactions between artists to be more immediate and personal. Social gatherings such as local Linux user groups and the well known dorkbot meetings all serve a similar purpose to OpenLab.

Like other FLOSS artistic collaborations, the advocacy of FLOSS movement is also one of its main objectives. However, it often has a less progressive approach in achieving this goal. OpenLab allows FLOSS ideologies to be propagated through its practice. Instead of strategically promoting FLOSS so that its effort can be maximized, it focuses on setting up examples and providing helpful environment to attract other practitioners. As a result, the effects achieved are comparatively subtle and informal.

Currently, there are approximately fifteen London based artists actively involved in the organization of OpenLab. The main methods of communication, besides

physical meetings, happen on the IRC channel and the mailing list. At the time of this writing, there are ninety subscribers on its mailing list. The topics on the mailing list range from practical arrangements to discussions concerning various technical and conceptual issues. OpenLab has also made alliances with other local groups and organization which share similar interests and objectives. For example, OpenLab has, from the very beginning, been associated with the dorkbot meeting in London. Besides these local groups, OpenLab also enjoys collaborating with other collectives that are based remotely. Goto10 for instance, has always been particularly close to OpenLab and has co-produced events in the past.

In 2006, OpenLab branched out to other cities and countries such as Glasgow and Portugal. Attracted by the success in London, these new OpenLabs attempted to apply the same ideologies to other remote locations. Whether or not these undertakings will achieve the same result yet remains to be seen; however it proves to a certain extent, that its original concepts and philosophy also met the needs of other practitioners. In late 2006, one more addition to the existing OpenLabs was proposed and began to take shape near the city of Milan, Italy.

One particular aspect of OpenLab that has further progressed recently is its autonomy. In the beginning, most of the arrangements and event proposals were predominantly carried out by a few members. As more artists became involved, further events and meetings have been organized by other members taking initiatives. In other words, OpenLab started to further fulfill its original goal, which was allowing dialogs and collaborations between its members to be freely and mutually formed without centralized leadership. As a result, the collective has become more democratic through a self-organizing manner. This transition is significant as it further increases the sustainability of the collective's practice and ideologies.

The last substantial OpenLab event, openlab#3, took place in a south London gallery in November 2006. This was a week long exhibition with two performance evenings on the opening and closing nights. It was the first time OpenLab curated an exhibition of such length. As a collective, OpenLab presented

a considerable number of works for the event - all created with FLOSS - and generated considerable interest from local residents and artists. Alongside this, the event also saw a number of new artists taking part in the exhibition and throughout the organization. These new artists have later become active members in the collective. It is particularly encouraging for a growing community like OpenLab to see fellow practitioners taking keen interests in its work and choosing to join forces with it.

3.2.3 Dyne

Dyne[3] is a long running artistic FLOSS collective founded by Denis Rojo in 2000. It became a registered non-profit organization in 2005, which formalized its activities. Rather than a group of individual practitioners, Dyne consists of several smaller groups predominantly based in southern Europe. Members of Dyne are deeply rooted in the hacker culture and can be traced back to various local hacklabs across Italy and Spain. The FreekNet Medialab in Catania, LOA hacklab in Milan, Metro Olografix in Pescara etc. are some entities involved in Dyne. Dyne itself is not based on a specific physical location - it consists of a network of artists and programmers who share the same ideologies and collaborate in cyberspace.

One aspect which sets Dyne apart from its peer groups is its strong political awareness. Its campaign for FLOSS in the context of art and media is largely derived from the social and philosophical implications of freedoms in software, rather than based on a practical and economical reasoning. For this reason, it can be said that its approach has been heavily influenced by pioneering organizations such as the Free Software Foundation. In other words, free software and the right to freedom are valued as a means to achieve a higher ethical order in a technologically driven society. This similarity in beliefs has consequently led to a close alliance between the two, and they have collaborated on many occasions.

Dyne's mission statement is described by its founder as the following: *"Dyne in-*

[3]http://dyne.org

tends to promote the idea and practice of open source knowledge sharing within civil society by fostering research, development, production and distribution of FLOSS based solutions: by opening the participation to online and physical communities, leveraging democratic and horizontal access to technology and lowering economic requirements for accessibility.". To achieve this goal, Dyne clearly defined three objectives:

- *To produce software that runs faster and better on old computers, as the possibility to recycle hardware is an important ecological issue we claim legitimacy for all possible tweaking of electronic devices existing.*
- *To foster use of FLOSS in artistic creation: exploring new forms of expression and interaction, disseminating new languages that can be freely adopted and modified, and ensuring everyone the long term conservation of digital artworks.*
- *To ensure sustainability of FLOSS development especially for non-profits. Since software is a socially relevant media, it should not survive solely on the basis of merchantability.*

Some interesting issues can be identified from the objectives above. The recognition of outdated hardware as an ecological concern reflects strongly to the "grass roots" approach of hacker culture. This hugely contrasts with many conventional media organizations whose goal is to implement cutting-edge and often inaccessible equipment. Dyne, therefore, consciously decouples the general misconception between innovation and the use of latest technology. It believes that the natural habitat for innovation lies in the wide dissemination and availability of software tools. In other words, if a given program can be used more often, it becomes more likely that it will be applied creatively to produce novel works. Furthermore, the emphasis on various sociological subjects such as the sustainability of FLOSS and the claim for legitimate hardware modifications, highlights the activist undertone of the organization. This constitutes the group's vigorous and proactive character in following its principles. Like FSF, Dyne believes that FLOSS is not merely an effective software development model, but could fundamentally benefit modern society in general.

The most recognized contribution of Dyne lies in its software projects. Over the years, it has produced a number of programs that have been deployed in an extensive range of contexts. "HasciiCam" marks the first software project of Dyne. It allows users to stream real-time video in the form of ASCII characters. Artistic novelty aside, it has proved to be practical for streaming visual images when bandwidth is limited. Under the same theme as media broadcasting, "MuSE" is a program that mixes, encodes and streams sounds. Its specifically aimed to help independent Internet broadcast to provide a practical and user friendly solution. The most notable project of Dyne is perhaps the "dyne:bolic" Linux distribution. Now in its second generation, dyne:bolic is a portable and versatile GNU/Linux operating system that meets the demands of media activists and artists. It provides some of the essential tools for multimedia manipulation broadcasting, whilst keeping the system easy to use and supporting a wide range of hardware.

Through its software projects, Dyne has accomplished a high degree of achievements. Users of its software include artists, broadcasters, journalists, researchers and activists. As a result, works of Dyne can be seen in places such as art exhibitions/performances, educational workshops and Indymedia events[4]. Dyne:bolic, in particular, has been hugely successful, and is estimated to have distributed over five hundred thousands physical copies world wide and has around five thousands visits a day on its web site. From FLOSS technical reviews to mainstream music production, dyne:bolic is frequently mentioned and featured. Moreover, Denis Rojo has been working extensively with the Netherlands Media Art Institute (NMAI), producing software solutions for video streaming and manipulation. as a result, NMAI is now becoming a formal patron of Dyne, which further ensures the sustainability of the collective's practice and accomplishment.

Software development aside, Dyne also serves as a platform on which collabo-

[4]The Independent Media Center (a/k/a, Indymedia and IMC) is a global network of individuals that consider themselves to be independent journalists and an alternative media outlet which takes a generally left-wing perspective on political and social issues. It is closely associated with the global justice movement, which criticizes neoliberalism, NAFTA, and the World Trade Organization. Indymedia is centered on an open publishing process that allows anybody to contribute.

rations can take place. This is clearly shown through the activity found on its web portal. These events also represent the rich cultural dynamics within its community. Hence, Dyne is not only a successful provider for innovative and accessible technology, it is also an educational and artistic collaboration.

3.2.4 Bek

Bek[5] is a regional center for electronic arts located in Bergen, Norway. It is also a non-profit organization providing essential services to the local artistic community. Unlike other artistic collectives mentioned previously, Bek has a physical location which consists of workshop spaces, studios and other facilities for artists to make use of. Additionally, Bek also offers various Internet-related services to its members such as web hosting and mailing-list.

Although Bek predominantly accommodates local practitioners with a sustainable working environment, it also frequently undertakes international collaborations with artists abroad. In other words, there are two strands in Bek's main activities. The first one provides a social space with necessary means, free of charge, for local artists to produce their work. This aspect of Bek is particularly popular amongst young artists and art students in Bergen. The kinds of creative practices which can be seen in this part of Bek, are mostly multi-media, technology related art disciplines ranging from digital sound/image manipulation to customized electronic installations. The second aspect of Bek's activity is an annual FLOSS digital art festival held in Bergen, which attracts a network of active artists, programmers and hackers to showcase their works and ideas. This has resulted in the flourishing of an artistic community derived from this festival, with many initiatives taking place virtually. In this context, Bek serves as a central hub for remote artists and their efforts to converge. Projects and discussions are co-ordinated through the festival and the community behind it.

Whilst the first type of Bek's activities does not necessarily focus on advocating the FLOSS movement, the latter part, has eagerly and pro-actively taken a stance in the promotion of FLOSS ideologies in the digital arts context. In

[5]http://www.bek.no/

this virtual community, led by Bek, many software projects have been suggested and produced. In the last few years, the main focus of this collective has been to develop a series of software solutions specifically aimed to address issues surrounding real-time video manipulation. Compatibility between existing software, unification of video specification and video streaming methods are some of the key topics that are being tackled. LiViDO[6] (Linux Video Dynamic Objects), for instance, is a video application programming interface that aims to provide simple method to create video processing plug-ins which can be used by multiple software packages. VideoPiping[7], on the other hand, was created to enable the supported software to share video data thus allowing them to network and achieve complex results. As this collective consists of several contemporary leading figures in video processing software development in Europe, the collaborations undertaken by these artists and programmers are likely to significantly determine and set the standard for future developments in the field.

One unique aspect for Bek is that it is supported by the Norwegian Council for Cultural Affairs and the Municipality of Bergen. This has given Bek vital stability in its operation. Such sponsorship is crucial in keeping Bek's practice sustainable, not only for its internal organization but also for projects and initiatives depending on it. For this reason, Bek has been a dominant and stable force in the field of FLOSS digital arts in northern Europe.

3.2.5 Mediashed

Mediashed[8], an organization led by a group of media artists aiming to encourage the practice of "free-media", was the first collective of this type to emerge in eastern England. It was initiated by Mongrel, an internationally recognized digital arts group, in 2005. From Mediashed's perspective, free-media encompasses a variety of contexts ranging from recycled hardware, publicly available information and FLOSS. In other words, any accessible mediums which could empower individuals' creative freedom are advocated. Furthermore, Mediashed

[6] http://www.piksel.org/Livido
[7] http://www.piksel.no/pwiki/VideoPiping
[8] http://www.mediashed.org/

has clearly defined the basic principles and ideologies behind free-media:

- *Its a freebie - doesn't cost much, especially to the people who need it most and can afford it the least.*
- *Its a low cost tool because it makes use of public domain Free and Open Source Software (FOSS) and recycles freely available old equipment, waste materials and junk.*
- *Setting one free by giving the ability and confidence to do things for oneself, promoting independence, self-reliance and the exercising of free choice.*
- *It is media that is open, transparent, unrestricted and outside proprietary controls, so one can freely change it, rewrite it or rebuild it to suit oneself.*
- *Supporting the free exchange of ideas and opinions to help build a more democratic society.*
- *Free and available to help when one needs support, advice, is looking for someone to collaborate with or just an informal discussion.*

Unlike other FLOSS groups which associate "free" with being the freedom in software, it is clear that Mediashed relates to such a term in several different ways. The low cost interpretation is particularly noteworthy. As the majority of Mediashed's work focuses on the underprivileged areas within the local community, economical influences are therefore a significant factor. In other words, a freely obtained illegal copy of a popular proprietary software may be more valuable and useful to individuals than FLOSS that is technically alienating. Mediashed actively acknowledges this circumstance and aims to present FLOSS and related knowledge in a accessible manner.

Conventionally, the low cost economical aspect of FLOSS is seen as the byproduct of its liberal characteristics in software development, and is therefore not a primary concern. Richard Stallman has addressed the distinctions between freedom in software and its economical costs on numerous occasions. However, despite FLOSS's general emphasis on the principles of freedom in software, the

first and often most obvious effect encountered by new users of FLOSS often is derived from the economical advantages. Thus, Mediashed shows its pragmatic approach by consciously embracing both aspects - philosophical and practical - of FLOSS.

Economic interpretations aside, Mediashed also identifies "free" at a more personal level. As technologies become evermore complex, they are often incomprehensible to the majority of the public whose daily lives are already reliant on them. This scenario is problematic, as individuals retain less and less freedom against imposing technology. By becoming more knowledgeable, one may gain further independence and freedom of expression.

One of the prime examples demonstrating the free-media practice of Mediashed is entitled "Video Sniffin". In this project, local youth were taught with the skills to make inexpensive electronic devices which allowing them to gain access to video signals transmitted by the local wireless CCTV cameras. As a result, they were able to locate cameras which could be later used for producing scripted films. With these devices, the local youth were able to experiment with accessible technology, whilst allowing them to better articulate their opinions through the footage produced.

Mediashed's activities include producing workshops, project hosting and facilitating its members with necessary means to realize their ideas. It pays particular attention to its relationship with the public and its contribution to the local community. This is made evident by the keen interest in engaging younger members of the public to allow them to discover their surroundings through free-media, thus improving the quality of life in the area. Furthermore, many of Mediashed's projects examine various political issues in public technologies such as wireless Internet connection and surveillance infrastructure by applying them in a creative context.

The practical approach of Mediashed can also be noticed in its strong bond with other regional collectives such as the Linux user group and other local societies which focus on a wide range of activities from poetry to music. By joining forces, these network of collectives form a strong voice in the cultural

development of the area.

3.2.6 Folly

Folly[9] is a not-for-profit digital arts organization located in north England. It provides a wide range of services both locally and on-line. Through producing workshops, exhibitions and performances, Folly engages with both regional artists and local residencies. It also has a web portal which hosts a variety of content from interactive Internet art, media broadcasting to knowledge based blogs.

Like other organizations, the creative use of technology in a sustainable manner is central to its ideology. Folly therefore materializes this principle through its digital arts related consultancy services to individuals and independent businesses. Naturally, FLOSS plays an vital part in the solutions it offer to its clients. In other words, Folly delivers FLOSS as an effective and empowering means to support not only creative activities, but also daily administrative operations within emerging artistic practices. As a result, areas such as project management, infrastructure design and web hosting are included in its consultancy service.

3.2.7 Access Space

Established in 2000, Access Space[10] is the first free media lab in UK. Its success is largely due to the unique combination of employing FLOSS and local recycled computers to achieve the necessary technical infrastructure for it to be operational. Using this model, Access Space has set a prime example of building a sustainable and resourceful community without raising any capital for hardware and software.

Access Space consists of a physical location in which anyone is free to visit and take part in its activities. Furthermore, participants are encouraged to

[9] http://folly.co.uk

[10] http://access-space.org/

take initiatives and make use of the equipment available to them. From simply making use of the computers to proposing projects and workshops, Access Space is very much led by its participants in a self organizing manner. Physical location aside, Access space also provide hosting services to artists and enthusiasts. To date, it hosts over two hundreds web domains, which consists of more than seventeen thousand on-line documents.

Currently in its seventh year, Access Space remains an active and valuable contributor to FLOSS and digital arts. For instance, it will host a live coding conference and festival in July 2007. It will be the first event of its kind in UK and possibly worldwide. The conference [11] will invite many of the renowned artists in the field to present and demonstrate their work.

3.2.8 Summary

The artistic collectives mentioned above represent an active development in FLOSS digital arts. These independent groups consequently form an interconnected social network between them, in which expertise and resources are frequently combined and shared. Such a model of collaboration has enabled them to work closely with one and another, thus further perpetuating the movement as a whole. Furthermore, they have also provided successful examples for artists who are keen to adopt FLOSS in their creative practice. For this reason, an increasing number of new media labs and groups following this ideology have been established.

The manifestation and benefit of FLOSS ideologies may be seen on many levels amongst these digital art groups. As individual artists, the collective environment provides the opportunity to exchange ideas and skills, allowing them to strengthen their creative practices. For instance, critical feedback through peer review is invaluable for artists not only in examining their works objectively, but also in relating their practice and views to others. The FLOSS principles are also materialized in the open relationships between organizations, enabling them to work together, rather than against each other, to contribute to the

[11] http://livecode.access-space.org/

advances in FLOSS digital art. For this reason, groups of different nature and characteristics are able to work together effectively, and fully utilize their individual abilities. Moreover, self-organizing characteristics can often be observed in the coordination amongst these groups, thus further reflecting the original FLOSS culture.

The emergence of these groups also highlights several noteworthy factors. The geographical origin of these collectives meant that the combination of FLOSS and digital art is not only established but is also indeed becoming an emerging practice throughout Europe. The examples presented above include the UK and a large part of Europe ranging from the north to south. This geographical wide spread of ideologies and aesthetics is remarkable, given that the FLOSS art movement is still young compared to other contemporary artistic disciplines.

Moreover, there is a wide range of contrasting characteristics and approaches between these groups. For example, some of them exist entirely in the form of virtual communications, whilst others have strong physical bonds to the local community they belong to. The methods used in promoting their ideologies also hold some significant differences. On one hand, some collectives adopted a casual and relaxed approach, whereas many others have a more formal and, at times, more political voice.

This contrast of characteristics is particularly apparent between Goto10 and OpenLab, which makes them an interesting comparison. Many aspects of Goto10's organization for instance, are more rigorous then OpenLab's. Issues such as group objectives, internal communications and project planning are generally precisely defined and thus its members have clear roles and focus within the community. OpenLab however, is somewhat spontaneous, which largely depends on its members making initiatives to manage various aspects of the group. As a result, Goto10 is typically more strategic and eager to achieve its goals, whereas OpenLab is inclined towards a rather moderate approach. One contributing factor to this difference may lie in the very nature of these two groups. Goto10 aims to promote FLOSS ideologies through the works it produces and supports. For this reason, many aspects of its organization have to be carefully thought through and executed, so that works produced can be exposed to a

wide audience. OpenLab, on the other hand, intends to advocate free software by creating social occasions which follow and practise the principles of FLOSS, thus providing a mutual environment for participants to communicate with each other.

However, the contrasting qualities of these two groups in many ways complement each other's activities. Whilst Goto10 focuses on the development of its projects, OpenLab provides a casual and informal context in which initial interests and feedback may be obtained. Furthermore, the emphasis on a local artists' community in OpenLab allows Goto10 to utilize its regional resources for activities such as workshops. As a result, these two collectives have enjoyed a close working relationship since 2004. This demonstrates how FLOSS artist groups, despite their differences, can collaborate and develop further.

Political awareness is another diverse attribute in these collectives. Groups such as Dyne and Mediashed consciously address several sociological issues and aim to improve existing cultural circumstances with necessary changes. This approach has much in common with other contemporary activist movements with different ethical concerns. As a result, these FLOSS artistic groups are often closely associated with developments in media activism, and at times, identify themselves in a broader civil context. Other groups, on the other hand, tend to adopt a more neutral political stance, where their practice emphasizes artistic merits and issues regarding the related aesthetics. These collectives delve into the area of creative innovations which would extend the boundaries of digital arts even further. In other words, the starting point of their practice lies in their artistic concern, rather than on a sociological perspective.

There are certainly many more other emerging collaborative groups investigating new territories in the domain of FLOSS digital art. Existing collectives are also eagerly exploring other forms of practices such as hardware design and literature publishing. Whilst they contributed to the development of digital arts and FLOSS movements with fresh perspectives, many of the early pioneering groups who influenced the current climate of the field continued to provide a valuable input and were recognized for their accomplishment. It is therefore foreseeable that a culture of collaborative groups will carry on playing a significant part in

the evolution of FLOSS digital arts, whilst yielding fruitful results.

3.3 Projects

As in any other creative practice, several projects have become successful and widespread. This section will aim to present a selection of examples to provide an insight into the ongoing progress in FLOSS digital art. Some featured projects were selected because they provide essential environments and tools for artists to create their work. This is analogical to the relationship between GNU/Linux, as an operating system and platform, and many other FLOSS software which were built upon it. Other projects chosen for this section are based on their innovation and artistic brilliance.

3.3.1 Pure Data

As mentioned in the previous section, Pure Data(Pd)[12] is a graphical programming language that allows artists to create complex customized programs, by visually arranging and connecting a collection of objects on the screen. These objects represent certain low level functions or operations, and thus by combining different objects, one can design algorithms which fulfill desired tasks. In Pure Data terminology, a program written in Pd is called a "patch", and the process of programming is therefore referred to as "patching".

Pd was originally designed as a sound manipulation software and was used to demonstrate the fundamental principles of real-time digital audio processing in the academic context[13]. As Pd is released as an open source software[14] and has an appealing graphical nature, it began to attract the attention of some artists and programmers. These early artists quickly joined the development of Pd and significantly extended its functionality. Over the years, Pd progressed

[12] http://www.puredata.info

[13] The book 'Theory and Techniques of Electronic Music'[29] written by Miller Puckette, the original author of Pure Data, uses Pd exhaustively to demonstrate sound processing and synthesis in the digital domain.

[14] Pd is released under the BSD License, http://en.wikipedia.org/wiki/BSD_license

from being an educational tool[15] to a fully-fledged multimedia programming environment.

Pd now specializes in many media related tasks from video processing to hardware interaction. Pd is currently one of the most used software tools in the field of digital arts and has a large user base.

One can easily understand the reasons behind the success of Pd. Since the popularization of digital arts, an increasing number of artists, who often had a background in traditional creative disciplines, were eager to experiment with the new medium and technology. For these practitioners, the transition required to obtain the necessary skills and to fully appreciate the new possibilities brought by computer programming, remained difficult and challenging. This was largely due to the fact that traditional art disciplines often have very tangible forms of practice. However, creating works of art through programming is entirely abstract, as it is based on the organization of logical expressions, rather than on the physical act of mixing paints or playing musical instruments, for instance. While most of the programming environments force artists to make such demanding adjustments, the graphical nature of Pd gives the underlying abstract operations in programming a far more tangible form and representation. As a result, artists have often felt that the immense learning curve inherent in programming is greatly reduced through using Pd. In other words, artists can typically learn and be proficient with programing in Pd in a short period of time.

For artists who already have an extensive knowledge of programming, Pd is also an attractive choice of software. The ability to customize Pd and to easily create additional functionalities enables technically inclined practitioners to adopt Pd to their specific demand. Furthermore, these changes and extensions can then be utilized and tested by other artists who use Pd in a more conventional manner. Thus, the flexibility and the re-usability of customizations in Pd offers an great incentive for advanced artist programmers.

[15]The primary target audience of Pure Data after Miller Puckett left IRCAM and subsequently released Pd as open source was for university students to learn and explore digital audio signal processing.

Because of its popularity, many artists and programmers have contributed to expanding Pd's capabilities. The most well known extensions include: GEM (Graphics Environment for Multimedia) for creating visual animations with the OpenGL graphics engine, PDP (PureDataPacket) for real-time video processing, PiDiP (PiDiP Is Definitely In Pieces) for adding extra capabilities to PDP, GridFlow for multidimensional dataflow processing in interactive video and pmpd/msd (Physical Modeling for Pure Data/Mass Spring Damping) for emulating various physical properties of the real world. Other external libraries also cover tasks such as network connectivity and hardware input/output methods. In short, Pd is one of the most comprehensive software tools which meets the needs of today's artists, whilst remaining user-friendly for newcomers. The versatile quality of Pd has also greatly contributed to its success.

Pure Data, therefore, has been employed in a wide spectrum of contexts in digital art. One of the most common usages of Pd is for real-time live audio/visual performance. In this genre, artists develop "virtual instruments" in the form of Pd patches. Through controlling and interacting with their patches in real-time, artists are able to demonstrate their works and give performances. Furthermore, as Pd allows different computers to easily communicate over the network, one of the emerging trend in this mode of practice lies in the domain of group improvised performance. In other words, through sending each other a series of messages, artists are able to synchronize and co-ordinate aspects of the performance as it progresses. Rhythmic timing and overall structure are some common attributes which can be altered and determined collectively during the performance.

Another popular use of Pure Data lies in the realm of multi-media installations. Typically speaking, the production of such works demands careful and precise integration of many media related elements. Hardware input/output and data interpretation/manipulation are a few examples needing to be designed and implemented. As this type of integrated capabilities is one of the strengths of Pd, many artists have employed it as the tool of choice to produce their work. In addition, as Pd supports an extensive range of hardware and operating systems, this also makes it a practical solution. In other words, artists are able

to employ Pd in almost any hardware and computers they may have, without specific requirements.

Lastly, Pd has also been used in media streaming over the Internet. As high bandwidth Internet connections are now becoming readily available, many artists are exploring its potential in the creative context. For instance, through streaming, artists are able to provide a more flexible and instantaneous access to their work. The distributed, decentralized nature of streaming also brought forward the possibilities enabling artists to create works of art which evolve within a network of intelligent programs.

Due to the widespread use of Pd, a great number of workshops providing essential trainings for artists, can now be found worldwide. Goto10, for example, produced a two week long intensive workshop in London focusing on various aspects of Pd in July 2006. This workshop, entitled 'Pd summer school', was supported by the Arts Council England. Many other cities in the worldwide are also home to vivid actives in Pd. The Institute for Electronic Music and Acoustics(IEM) in Graz has integrated Pd into many of its research projects, and hosted the first International Pure Data Convention in 2004. They consequently published the proceeding containing several critical writings on the subject entitled "Bang". Pd also has an active community in Canada, particularly in the region of Montreal. Together with several local artistic and educational organizations, they are currently planning for the second International Pure Data Convention, which will take place in August, 2007. New York and Barcelona are also cities that have witnessed the growing user community and practices of Pure Data. As it becomes ever more popular, its functionalities and capacity as tools will also further expand. This is truly exciting for artists who uses Pure Data as a creative tool.

3.3.2 SuperCollider

SuperCollider(SC)[16] is a programming language for audio synthesis and manipulation, originally developed solely for the Macintosh operating system. It was

[16]http://www.audiosynth.com/

also created to explore the domain in algorithmic composition. Like Pure Data, SuperCollider offers flexible and extensive functionalities which enable artists to create complex works of art. However, SC is a text-based programming environment where artists develop their program through writing lines of codes. Without a graphical user interface, SC is often considered harder to learn and be proficient with. Nevertheless, SC has several significant advantages by favoring text over the graphical alternative.

Despite the initial learning curve, expressing logical designs in the form of text, to a large extent, is far more effective than the graphical approach. At first sight, editing lines of text is a rapid process in comparison with the conventional mouse driven input method in any GUI[17]. As a result, although it may seem less intuitive at the beginning, programming with text is efficient and productive. Without the additional visualization in programming, it may be argued that artists are forced to focus on the fundamental abstract structure of the algorithms, which could allow a higher degree of rigor and sophistication to emerge in the end result. In other words, the process of programming is free from external factors and influences brought by the intermediate graphical representation of code. Therefore, these characteristics in text-based programming have attracted many artists to prefer SC over other tools which have an "intuitive" user interface.

SuperCollider was not a FLOSS project until the release of its third version(SC3), where GPL was chosen as the license. SC3 also represents an important departure from its previous versions in its rudimentary design and technical implementation. One of these changes lies in the separation of SC into a server and a client. The client is dedicated to the interpretation of the SC language and sends instructions to the server. According to the transmitted messages, the server will perform the necessary computation to synthesize or manipulate sounds. This change enabled SC to be used in a very flexible manner, since the client and server do not necessarily have to be run on the same computer. SC3 includes some fundamental changes to its language specifications and internal structures. For this reason, many artists who became familiar with the previous

[17]Sean McDirmid also pointed out similar comparisons between textual and visual programming languages in his paper 'Living it up with a Live Programming Language'[23]

versions found the transition to SC3 challenging.

SC3 has now become available for other operating systems. There is currently a good support for GNU/Linux and a beta version for Microsoft Windows. An interesting development in SC3 on GNU/Linux is the integration with the well known Emacs text editor. This means one can write and execute SC codes from Emacs, and control various aspects of server which are currently executing. As a result, using SC3 in this manner directly inherits Emacs's hugely comprehensive text processing features, which further increases the overall efficiency. Moreover, as Emacs have an immense user base and SC3 is the only Emacs mode[18] specializing in algorithmic music composition, SC3 could potentially increase its community by drawing users from Emacs.

Although SC has been used in a variety of compositional and performance contexts, one the most notable use is perhaps in the practice of live-coding. The "Just In Time"[19] library(JITLib) of SC is specifically designed to facilitate the rapid and direct modification of internal compositional/synthesis structures defined by artists. As mentioned previously, live-coding brings the process of programming in front of the audience, to better reflect on artists' creative conception and skills. This approach reexamines the nature of digital audio visual performance and hypothesizes an alternative relationship between artists, codes and the spectators. SC3 has thus become one of the most prominent tools for artists to explore the territory of live-coding.

At present, SuperCollider still remains an audio processing only environment, and has limited hardware support. Its current development nevertheless points to innovations at different levels. The PLT-Scheme[20] client, for instance, enables the SC server to be used and controlled with the language called Scheme. As this language is highly regarded for its minimalist philosophy, and is used in many experimental software projects, by adopting a standardized and effective

[18]Emacs uses different modes to alter the overall behavior of the program. Certain functionalities and commands are only activated when editing particular types of text files. For example, there are different modes for editing normal text, source codes of different programming language and more

[19]http://swiki.hfbk-hamburg.de:8888/MusicTechnology/566

[20]http://www.plt-scheme.org/

language in SC client, its server may be used in conjunction with other PLT-Scheme compatible software. This would further encourage the creative use of SC overall.

3.3.3 pure:dyne

pure:dyne[21] is a live GNU/Linux distribution optimized for the purpose of real-time audio and visual applications. Live distribution refers to a type of GNU/Linux operating system that can be run from either the CD alone, or by installing it inside of an existing system by simply copying the necessary files. For this reason, artists are able to take advantage of and use FLOSS without compromising their previous working environment. As its name suggests, pure:dyne is built upon the dyne:II[22] platform and originally optimized for Pure Data. However, pure:dyne now also contains several other interesting and useful creative software, and is becoming evermore practical to be used as a complete GNU/Linux distribution for both media art and daily tasks.

The development of pure:dyne can be traced back to the inclusion of Pure Data in the first dyne:bolic liveCD distribution [23]. As this addition later became increasingly popular, there was suddenly a demand to increase its support for Pure Data in a more serious production context. Meanwhile, the dyne:II core that Denis Rojo [24] had been developing for the forthcoming version of dyne:bolic provided the necessary development tools needed to make such a customized distribution for Pure Data. As a result, a collaborative effort began between dyne.org and Goto10 in early 2005 to work towards a distribution based on the new dyne:II core.

After a year of development, pure:dyne started taking shape and began its beta testing. In late 2006, pure:dyne officially left beta to have its first public release. Today, pure:dyne gathers a growing user community and has been used in numerous workshops, exhibitions and performances. Although nowadays

[21] http://puredyne.goto10.org
[22] dyne:II is platform in which a fully functional and portable system can be built using it.
[23] The first inclusion of Pure Data can be found in dyne:bolic1.4
[24] Founder of Rasta Software and the key maintainer of dyne:bolic

other multimedia oriented live GNU/Linux distributions may be found, many aspects of pure:dyne still remain unique amongst them.

One of the most important aspects of pure:dyne is that it attempts to offer both practical and portable solutions for practitioners in the fields of FLOSS digital art. This is because although many portable distributions are available, they are mostly used for demonstration purposes, and are thus not suitable for professional production. pure:dyne, on the other hand, allows artists to create extensive works whilst keeping the entire system, including artists' works, very portable. This makes it an attractive alternative for artists who wish to develop projects but do not have access to a dedicated environment.

Moreover, accessibility is also an important part of pure:dyne. pure:dyne recognizes artists who intend to take advantage of the innovations in FLOSS in the creative context but do not have the resources and abilities to complete the lengthy installation, configuration and even compilation of the required software. Because of this, pure:dyne aims to provide a functional and highly optimized environment which requires a little learning curve.

Lastly, pure:dyne follows a minimalist approach in system setup. This enables it to be more streamlined and "clutter free". For example, the default desktop environment is FluxBox[25] as window manager and applications such as Rox-Filer[26] and Xfe[27] can be used for supporting the conventional representation of files and directories. pure:dyne also includes window managers such as ratpoison, evilwm and dwm. Such an approach enables users to achieve greater productivity when using the system.

There are two important elements in pure:dyne that enable it to achieve its objectives. These two components the dock and the nest.

- **Dock** - A dock refers to an "installation" of pure:dyne onto the host system. A dock contains all necessary components that are required to

[25]FluxBox is a minimalistic window manager for X11. http://fluxbox.sourceforge.net/

[26]Rox-Filer provides the conventional desktop representation of icons and files. http://rox.sourceforge.net/desktop/static.html

[27]Xfe is a file browser, similar to the file explorer found in Microsoft Windows. http://roland65.free.fr/xfe/

boot pure:dyne entirely from the storage device. The process of docking is extremely straightforward, it only requires copying the /dyne directory from the CD or ISO image onto a partition readable [28] by pure:dyne.

- **Nest** - A nest (.nst) is a file that a user can create once pure:dyne has successfully booted. This file contains a user's home directory and configuration files [29]. The nest file can be stored on either the hard disk or on a portable storage device such as a usb key. During the boot process, pure:dyne will look for the nest in any of the partitions it finds and mounts the nest at the appropriate location. Through the integration of UnionFS [30], users can easily save and store any modifications made on the system.

With the further development of the dock and the nest in dyne:II [31], pure:dyne can be used with a greater flexibility. For example, a system running from a CD or hard disk, in combination with a portable storage device will result in a complete functional system. Once the system is successfully booted, a user can simply write to his or her own home directory and continue working the same way regardless of which storage device is being used. One other obvious advantage of the docking system is that pure:dyne can co-exist with other operating systems in a very straight forward manner, as all pure:dyne related files are contained in one single directory. Updating to a newer version of pure:dyne only requires to overwrite the content of the dyne directory. Lastly, by simply creating new users following the conventional GNU/Linux method, a nest can also support a multiple user system.

dyne:II also has a modular system in which applications may be packaged and distributed. Each package is a compressed [32] .dyne file which is stored in the designated directory. For instance, the applications included in pure:dyne exist

[28] current supported file-systems are: fat vfat msdos ntfs ufs befs xfs reiserfs hfsplus ext2 ext3

[29] A nest contains the /home, /root, /var, /tmp and /usr/local.

[30] UnionFS allows transparent overlay of files and directory from different file-systems. http://www.unionfs.org

[31] Both dock and nest existed in dyne:I. However, these two elements were significantly further developed in dyne:II.

[32] .dyne modules use the squashfs read-only file-system. http://squashfs.sourceforge.net

as a module, named pure.dyne, of dyne:II. This means that users and developers can simply package their favorite applications [33] and swap between them. To include a new module, one simply needs to copy the .dyne file into the /dyne/modules directory and either reboot or mount the module directly.

In short, pure:dyne may be used/installed in the following ways:

- Used with the CD alone, without saving user data.
- Used with the CD in conjunction with a portable storage device containing the nest.
- Used with a dock on the hard disk plus a nest either on the hard disk or a portable storage device.
- Used with both the dock and the nest on the portable storage device. for example, running pure:dyne entirely from solid state memory.

3.3.4 PacketForth

Influenced by Forth, a procedural and stack-oriented programing language invented in early 1970s, PacketForth(PF)[34] is a programing environment for multimedia processing. PF is suitable for a wide range of tasks from prototyping to academic research and professional production. This is mostly due to the characteristics of the language. The simplistic syntax of PF not only makes it quick to learn and program, it also enables artists to rapidly sketch their ideas and see the effects immediately. Furthermore, as PF allows users to effectively define their own set of functions and vocabulary, this encourages artists to build extensive systems and investigate their properties methodically.

One principle central to the design of PF is the communication and relationship between artists and the program. The complexity of software often reflects the increasing capability in computing hardware. As a result, the transformation

[33] Currently there are modules for Ardour, network tools, Gimp, OpenOffice, BitTorrent, dvd authoring and more.

[34] http://packets.goto10.org

from the initial conception, through code and program, to the final works of art is lengthened due to the additional software components and layers. This arguably can misdirect artists' attention and abstract their creative process, instead of helping them to utilize the full potential of the system. PF, therefore, aims to provide a fluid and minimalistic environment in which the process of programming is analogical to converse with the system. For this reason, every aspect of the environment can be rapidly defined and dynamically modified, truly respecting the open and transparent exchange of information between artists and the system.

The open ideology also reflects to how PF can be used. First, it may be used interactively with users entering PF commands sequentially. Secondly, one could write a collection of commands as a PF script, which is to be executed at once. These two methods may be combined freely, with artists loading a collection of PF scripts and interactively altering their parameters and behaviors. Additionally, PF also allows it to be easily connected to other programs. PF could interact with Pd through creating PF scripts emulating the functionalities of a Pd object. One could also execute standard Unix commands within PF, to interact with the operating system through various system tools. With the implementation of OSC[35], PF is able to effectively exchange data with other OSC supported software. Lastly, in a similar way to SuperCollider, PF can also be controlled within Emacs, which also inherits many of its capabilities. This versatility is therefore also one of the main goals of PF, which aims to develop it into a 'media glue language', where many different media programing paradigms can work as one.

Although PF is a relatively young software, it has gained wider interest and has a growing user community. Even though its design and usage may seem abstract at first, PF has been particularly successful in workshops, which are aimed to introduce computer graphics in the creative context. In other words, workshop participants with a typically limited programming experience, have responded with great enthusiasm and have found PF interesting to program. At present, the most extensive works produced with PF are perhaps a series of

[35]Open Sound Control is a communication protocol, it has been particularly successful amongst creative software tools. http://opensoundcontrol.org/

installations, entitled “Metabiosis”, exploring various phenomena of artificial life in the artistic context. Metabiosis is a joint project between two artists, Marloes de Valk and Aymeric Mansoux. The first series of the work was exhibited in late 2006 in the Netherlands Media Art Institute.

Since the release of PacketForth, its author has now developed a suite of software tools continuing the influences of Forth and the ideologies of PF. These tools are designed for programming micro-controllers, following the interactive and transparent approach seen in PF. Artists are thus able to write software for micro-controllers with immediate feedback bypassing the conventional lengthy process of source code compilation and debugging.

3.3.5 Fluxus

Fluxus[36] is one of the first software applications developed specifically to explore the practice of live-coding in the context of 3D animation. It consists of a 3D OpenGL[37] rendering engine and a scripting environment. It also provides many useful functionalities such as physical modeling, audio input analysis and OSC implementation, which can all be utilized to create complex and interactive visualizations. Fluxus pioneered a unique method in interactive programming by overlaying the codes with the visuals it generates. In other words, both the animation and its programming text editor are located in the same window. The implication of this feature is practical, as well as philosophical. On one hand, it enables the use of the software to be productive and economical, as artists are able to program and see its immediate effects without the constant switching of focus. On the other hand, by exposing the underlying code of the animation, it reflects the principles of live-coding: the process of programming and the resulting code should be transparent and accessible. These implications are particularly evident in the performance context.

The language used in Fluxus scripting is called scheme, and is a minimalistic language which supports many programming paradigms. It is also one of the two

[36] http://www.pawfal.org/Software/fluxus/

[37] OpenGL is a standard API for producing cross-platform 3D graphics. http://www.opengl.org/

main dialects of the well known Lisp[38] language. The simplistic nature of the scheme makes it effectively easy to learn and program in real-time. Moreover, it also allows complex programs to be rapidly developed through a collection of smaller functions, as they can be easily defined and used. In many ways, the reasons behind the use of scheme have much in common with the choice of Forth in PacketForth. These two languages are both economical in their syntax and elegantly designed.

As Fluxus is a programing environment for computer generated graphics, one interesting usage lies in building intermediate systems. For instance, a video game can be developed entirely within Fluxus. In fact, there are many examples which illustrate this category, such as a game-pad controlled live-coding system, entitled 'BetaBlocker'[39]. This shows that live-coding aside, Fluxus is an extensive tool which can also be used in a wider range of contexts such as data visualization or for educational purposes.

Fluxus's reputation amongst the digital art community has grown in recent years. It has appeared in a number of international festivals, and many workshops have been organized to help keen artists learn and use the program. Its success and innovation has inspired the production of "Flaxus" by a group of independent artists[40]. It is a flash based web application mimicking various aspects of Fluxus. Although the implementation of "Flaxus" remains debatable, especially with regards to its implications and contextualization, it nevertheless proves that the ideologies of Fluxus extend beyond its original intention.

3.3.6 Conclusion

There are certainly many more FLOSS software projects which are widely used and are being actively developed. Whilst the above selection tends to cover programming environments for producing works of art, there are other types of software projects which fulfill different purposes. Software projects aiming to

[38]Lisp is one of the oldest and most powerful high-level programming language. http://en.wikipedia.org/wiki/Lisp_programming_language

[39]http://www.pawfal.org/index.php?page=BetaBlocker

[40]Flaxus is by Ivan Ivanoff and Jose Jimenez. http://www.i2off.org/flaxus/

provide practical solutions to common creative tasks with a unified protocol and standard are one of them. Although these projects may seem less artistically oriented at times, they do contribute significantly to the overall innovation in FLOSS digital art, by strengthening and improving the infrastructure other programs rely on.

The 'Jack Connection Kit'[41] is a prime example project of this type. Jack is a low-latency audio server that allows all supported programs to route audio signals freely between them. As a result, each jack enabled program is no longer an isolated software, but belongs to a suite of tools which can be used collectively. Since signal routing is one of the most common tasks in audio processing, the solution offered by Jack is truly invaluable. It is now becoming the standardized implementation for handling applications' audio input and output. For example, all the above mentioned software have Jack support.

Another notable project in this category is the Open Sound Control(OSC)[42]. OSC is a protocol which specifies how data can be transmitted and understood between software over TCP/UDP connection. Through the use of OSC, different programs can communicate with one and another, thus making it easy to co-ordinate tasks and share data. In SuperCollider, OSC is adopted as the communication protocol between the client and the server, and because of this, any other programs able to follow these OSC conventions may be used to control the SC server. This plays a major part in the flexibility of SuperCollider. In fact, all programs in the selection allows OSC communication, which further extends the possibilities of how they may be used.

One other category of FLOSS projects worth mentioning, belongs to a more conventional creative environment. These software programs correspond to the simulation-based digital art practice identified in the previous chapter. They enable artists to create works with a more familiar interface, often modeled after corresponding physical instruments or environments. Ardour[43], for instance, is a multi channel hard disk audio recording software, which musicians can use to produce recorded music. Because it also supports Jack, Ardour may

[41] http://jackaudio.org/
[42] http://opensoundcontrol.org/
[43] http://ardour.org/

be used with other experimental software, such as Pure Data, to provide an extensive recording and editing environment. Similar to Ardour, Cinelerra[44] and LiVES[45], are non-linear video editing software in which video footage can be edited, processed and arranged to produce a final composition. In addition, LiVES can also be employed in a real-time context, allowing it to be used in visual performances. Lastly, Blender[46] is a 3D animation program that performs a wide range of tasks from modeling, rendering and compositing to creating interactive 3D applications. It is highly praised for its vast capabilities and has a considerable user community.

One of the latest emerging trends in the development of artistic FLOSS projects lies in the domain of open hardware. In these projects, the conventional source code to a program is substituted by the design and schematic of a given hardware. Artists, can therefore produce such devices according to the available information and modify its design if desired. Arduino[47] is currently perhaps one of the most successful open hardware projects. It is a circuit with a microcontroller which can be programmed to accept inputs from sensors, or control components such as lights and motors. It also consists of software environments which allow it to be a stand-alone device once completed, or to work in conjunction with other programms like Pure Data.

While the number of possible examples remains limited, this section nevertheless aimed to present a comprehensive collection of FLOSS projects which are both innovative and widely recognized in the digital art community. It also aimed to reflect to the current development of the ever evolving field of FLOSS digital art. Furthermore, these projects illustrate the contrasting approaches to software design found in FLOSS, with some maintaining a strong historical influences whilst others are cutting edge and innovative. For instance, Pure Data can be traced back to the the programming paradigm, generally referred to as Music-N[48], invented by computer music pioneer Max Mathews in the late 1950s. Fluxus, on the other hand, represents the new and emerging ideologies in digital

[44] http://cvs.cinelerra.org/
[45] http://lives.sourceforge.net/
[46] http://www.blender.org/
[47] http://arduino.cc/
[48] http://en.wikipedia.org/wiki/MUSIC-N

art. It is apparent that there is a rich variety of projects, all contributing to the FLOSS digital art movement. Although each project has its own focus, they are often designed to be compatible with one and another, so that they may be freely combined. This unique characteristic of FLOSS projects is extraordinary, as it allows the movement as a whole to further develop, whilst keeping a unified internal design and standards. From the practitioners' perspective, one can create works for art with ever increasing possibilities without compromising one's creative expression.

3.4 Events

Having witnessed the development of FLOSS digital arts in Europe, a number of events have been specifically curated to demonstrate and promote its practice. These events - mostly artistsic festival and academic conferences - are becoming vital outlets for individuals and groups to showcase their works. This section will therefore provide a selection of examples highlighting some key events focusing on FLOSS and digital arts.

3.4.1 Make Art

Curated by Goto10, Make Art[49] is an annual festival devoted to FLOSS digital art. Make Art first took place in Poitiers, France in 2006. Currently in its second year, Make Art was again held in April 2007 at the same location. The festival lasts a week, and consists of workshops, exhibitions, presentations, discussion panels and performances. Make art focuses on the disintegrating boundaries between art and software programming. The event is dedicated to artists who create their own tools, and apply the same rules to art as to FLOSS. It attempts to engage both invited practitioners and the local community with the effects of FLOSS in digital art.

The first Make Art festival was one of the largest events produced by Goto10. At this time, much of the focus was on forming partnerships with regional and

[49] http://makeart.goto10.org

national cultural institutions to support the event. As the FLOSS and community focus appealed to many local authorities, Goto10 gained much needed sponsorship from key organizations such as le Ministere de la Culture et de la Communication, la Region Poitou-Charentes and le Conseil General de la Vienne. The first Make Art also attracted media attention, and it was featured in a variety of press and radio: Liberation, a national newspaper, published a two page report on the event.

Following the initial success, Make Art 2007 witnessed several significant progresses. In terms of continued support from the local institutions, one of the Goto10 members was funded (since 2006) to produce the new edition of the festival. This crucial change was necessary, as organizing events of this scale requires much attention and co-ordination. Having a dedicated personnel thus ensures the production of the event is well thought out and executed. Make Art 2007 also adopted a "call for works" approach to give new artists opportunities to demonstrate their practices and be involved with the event. Goto10 received over seventy applications varying from performances to installations. The number of artists applying proved that, even only in its second year, Make Art met the demands of artists and started to gain a wider recognition. The new edition of Make Art featured a wider spectrum of art works, originating from internationally established artists to emerging independent ones.

As a result, the number of artists who were invited to Make Art 2007 was far higher than that of the first year. For instance, three concerts[50] (instead of one in 2006) were created, with each of which focusing a different genre of live performances. Furthermore, the scale of the exhibition also increased significantly, consisting five multi-media installations and the screening of a documentary on FLOSS entitled "The Code"[51]. As the main target audience of the festival is aimed at members of the general public, Make Art 2007 also employed a professional interpretation service to assist with real-time translation, via radio broadcast, throughout the presentations and panel discussion.

[50]The three focuses of these concerts were: live noise concert, audio/visual performances using physical modeling in Pd and general live performances using FLOSS

[51]http://www.code.linux.fi

In addition, Goto10 had also formed partnerships with other FLOSS groups - such as Bek - in the planning of Make Art. The aim is to provide opportunities not for both individuals and different organizations to collaborate. Thus, artistic FLOSS collectives can join their efforts, reinforcing the development of their practices.

3.4.2 Piksel

Piksel[52] is an annual convergence of FLOSS artists and programmers, who congregate to examine various current technical and philosophical issues in FLOSS digital art. First launched in 2003 by Bek, and held in Bergen, Norway, Piksel is one of the longest running events of this type. Many key figures in the field regularly attend the event, therefore contributing to the in-depth and engaging characteristic of the event.

The target audience of piksel is predominantly aimed towards existing practitioners in FLOSS. In other words, by limiting its participants, it is able to obtain a more focused and engaged atmosphere throughout the event. In addition, Bek also uses Piksel as an opportunity to discuss and report on many software projects it supports. For this reason, Piksel is generally regarded as a meeting place for developers and artists to co-ordinate their efforts and receive critical feedback from peers.

Over the years, Piksel has presented some of the most innovative projects and has addressed several current technical and philosophical issues in the field. Similarly to Make Art, it also actively seeks collaborations with other groups or events. Goto10, dyne.org and ap/xxxxx[53] are some of the collectives regularly taking part in Piksel. For instance, ap/xxxxx curated "XXXXX_AT_PIKSEL" seminar, as part of Piksel 2006. The central theme of this seminar was based

[52] http://www.piksel.org/

[53] ap/xxxxx (http://1010.co.uk/) is a collaboration between British artists Martin Howse and Jonathan Kemp. Their practice encompasses both theoretical and practical aspect of digital arts. Speculative hardware, code and free software are some of their main artistic concerns. Their latest publication, entitled 'xxxxx'[17], consists of a great number of critical writings contributed by both academic theoreticians and active practitioners. It offers and extensive discourse into the practice of digital arts.

on the development of Open Hardware, which is one of the emerging and critical subject in the digital art scene. This seminar also invited internationally renowned scientists Bruno Marchal and Otto Roessler, to lead the discussion on art, science and technology.

3.4.3 Linux Audio Conference

Linux Audio Conference(LAC) is a yearly academic conference presenting current audio related developments using open source software. It includes a wide range of subjects from music composition, audio production, device drivers to content distributions. LAC also features practical demonstrations and workshops, creating opportunities for participants to learn and experience different techniques and approaches concerning audio on FLOSS platform.

Currently in its fifth year, LAC has become the authoritative figure in the field. Its proceedings documents the up to date progress on many of the significant and cutting edge projects related to audio and computer music. As an example, LAC 2006 demonstrated wave field synthesis[54] at a remarkable scale consisting of two thousand wall mounted loud speakers, using a highly customized FLOSS system named sWONDER. Furthermore, various issues relating to firewire based external audio devices in Linux was also presented during the conference. As many of such devices have been produced and the current support is somewhat limited, the "FFADO"[55] project aims to bring generic and high quality implementations to allow these devices to function with Linux systems. Lastly, pure:dyne, a live GNU/Linux distribution optimized for real-time audio/visual applications, was also presented and demonstrated as part of LAC 2006.

3.4.4 International Pure Data convention

As mentioned previously, the Pure Data convention is an opportunity for its community to meet and collaborate. In 2004, the first ever convention took place

[54] WFS is used to reproduce the spacial location of a given sound within a set of loudspeakers. http://en.wikipedia.org/wiki/Wave_field_synthesis

[55] http://www.ffado.org/

in Graz and was organized by IEM. The event saw many of Pd's core developers, including the original author, presenting and discussing various aspects of Pd. The convention also represents an important social occasion for the community. As communication between the majority of developers and users usually takes place over the Internet, the event created a rare chance for many of them to finally meet face to face. More remarkably, the convention was one of the first to celebrate a creative FLOSS project of its kind in such scenario, and the event had a great attendance. This shows that Pure Data, as an artistic software environment, now has an user community extensive and resourceful enough to produce such an event.

The result of the first convention has also been published as a book[42], documenting the activities and debates occurring during the event. It includes interviews and theoretical papers, offering the first comprehensive literature on the insights into Pure Data.

Although not an annual event, the second convention was scheduled to take place in August 2007 in Montreal, Canada. Following the success of the first event, the second convention has been greatly anticipated by the community. Furthermore, the change of location also reflects to the global widespread of the software.

3.5 Conclusion

This chapter will have hopefully provided sufficient evidence on the evolving practice of FLOSS digital art. Although it is by no means a mainstream phenomenon, it is now gathering momentum whilst gaining a wider recognition in the public eye. The rapid development of FLOSS digital art is not an incidental event, but rather a convergence of two distinct movements which share many similarities. The marriage of FLOSS and contemporary digital arts has proved to be fruitful and produced a wealth of engaging and challenging results. Some of these effects are technological, others are sociological. If its success continues, one can foresee creative practices based on FLOSS ideologies becoming a common practice. This is a culture that embraces the freedom, not just to express,

but also to create and distribute works of art.

Chapter 4

Practical Project

4.1 Introduction

The artistic practice from which this thesis is derived from has experienced significant changes throughout the course of this research. On a technical level, it has moved away from conventional proprietary software to adopting FLOSS exclusively. Artistically, it has collaborated with many other innovative practitioners, who also share similar interests and concerns in digital art. Most importantly, it has taken some influences from the ideologies in FLOSS, to identify itself as a part of the open community sharing resources and works. As a result, the combination of these factors have in many ways broadened its horizon, allowing it to further evolve. Several key issues have emerged while these changes took place, thus providing the starting point of the practical work in this research.

As software and FLOSS became the central component of the practice, their influences on the creative processes were clearly experienced and observed. Because of this, the practice gained greater awareness of both the implicit and explicit effects of software in digital art. As a result, it particularly values software programs which are flexible and extensive in their functionalities, so that artists can externalize their conceptions freely. In other words, a software offer-

ing less limitations generally encourages a diverse artistic community to emerge who are open to innovations. Therefore, artists should strive to develop and design programs that reflect the open ideology found in FLOSS, where creative expressions are not restricted by predetermined functionalities and features.

Having adopted FLOSS and followed its principles, the practical work should aim to bring an original contribution to the artistic community. This not only acknowledges other's works on which the practice is based, but also allows it to take on an active role in the field. To achieve this, it must first be able to identify a specific area in which further progress can be made. By examining the current development and resulting shortfalls of the software that it relies on, it may reveal possible options for the practical work to be focused on.

At present, the research has largely been based on the understanding of FLOSS digital art from the user's perspective. In other words, the personal practice consists of employing FLOSS to produce creative works. However, in order to gain greater insights, it must also be involved in the development process. Thus taking part in developing software would contribute to a more representative and in-depth knowledge of its context. Moreover, this would also allow one's skill in programming to advance further.

4.2 Personal practice

The personal practice should now be presented to further establish and contextualize the practical work of this research. It has been predominantly focused on the realm of live audio performances[1], and pays particular attention to the generative aspects of the composition. Its primary interests lie in developing performance systems which are guided by the author in real-time to produce the final result. Therefore, a typical creative process involves constructing intermediate compositional components, and rehearsing with the overall system to determine the final structure and detail of a piece. Furthermore, the visual element in the performance is often accomplished by collaborating artists sharing similar ideologies.

[1] See Appendix C and D for further documentation

Pure Data was employed to create such compositional and performance systems. As the process of programming is often where composition takes place, Pure Data's intuitive visual representation of code makes it a particularly suitable tool. It allows compositional ideas to be prototyped and tested rapidly, whilst it also has sufficient mechanisms to allow extensive systems to be created. Furthermore, having a customizable graphical user interface allows the interaction between the user and Pd patches to be accessible and easily defined.

The use of Pure Data was also based on practical reasons. Prior to adopting FLOSS, Max/Msp was the prime choice of software and was extensively used in performances. Pd therefore became the natural candidate in the migration to FLOSS based tools. Given the great similarity between these two programs, it meant that the skills learned previously could be easily applied to the new environment. This significantly minimized the time and effort in the transition to a different software.

Having accumulated experience in performing with Pd, one of the recent focuses in the practice has been developing a modular performance environment. In other words, components of the system are programmed so that they can be easily reused and deployed. This results in a greater degree of flexibility in the compositional context, as modules can be freely combined and removed. Furthermore, it also aims to create generic modules according to the roles they may play within the system. This can potentially minimize the complexity in the programming, thus making the environment easier to maintain and further evolve.

Many performances have been created using this customized compositional system described above[2]. Furthermore, these performances took place in a wide variety of social contexts, including academic seminars, gallery exhibitions, festivals and club nights. Such a mix of environments provided a crucial and realistic assessment for the future development to be based on.

Another area which later became increasingly dominant in the personal practice is promoting the use and ideologies of FLOSS by teaching workshops. Pure

[2]See "selected performances", Appendix C

Data, in particular, has been the most common subject in these workshops. They typically involve the introduction of elementary Pure Data programming, helping participants to understand the scope of FLOSS as an emerging technology. The aim is therefore to demonstrate and provide alternative means for artists to materialize their works.

These workshops often provide an ideal backdrop for further collaboration to take place. This is mostly because participants are often active artistic practitioners who are keen to explore new possibilities. As participants each have unique practices and experiences, mutual communications can often be created and observed amongst them, with ideas and resources being exchanged. Being involved in workshops has thus resulted in hugely positive effects on my personal practice, not just by allowing personal knowledge to be formally articulated, but also by establishing communications with others in the same field.

While searching for suitable practical work in this research, it quickly became apparent that the focus should be based on Pure Data. The experiences gained through programming in Pd and teaching it in workshops would allow critical analysis on its current design and usage to be made. Furthermore, the analysis can also point out areas for improvements, which the practical aspect of the research could aim to address and resolve. As Pd is one of the most widely used programming environments in digital art, any resulting improvement would consequently be beneficial to the large community which relies on it, thus contributing to its future development.

As a result, the main objective of the practical work, is firstly to examine the present limitations of Pure Data, then subsequently to implement the proposed improvements.

4.3 Limitations of Pure Data

The graphical nature of Pure Data is undoubtedly the advantage over other software environments. The dataflow visualization of codes makes the process of programming generally an intuitive and rapid process. However, several

problematic areas may be found in its graphical user interface. Although these drawbacks have long been acknowledged by its community, significant improvements are yet to be made. The following section will attempt to summarize these issues.

4.3.1 Lack of customization

The graphical user interface of Pure Data has always had a minimalistic appearance. While this characteristic has certain practical advantages and, at times, aesthetic appeals, it however could offer greater flexibility from users' perspective. The ability to customize its appearance may seem to be derived from a decorative need, which is often valued as nonessential amongst some of its developers. However, with a closer look, having customizable user interface may have deeper effects than a simple cosmetic change. Although the visual appearance is generally not considered as a core functionality, it can nevertheless influence how Pure Data is adapted and used at many levels. Because of its simplistic nature, many new users are often put off by its graphical implementation. In other words, the first impression of Pure Data, by being inflexible, does not convey its capabilities as a creative tool. Moreover, as Pure Data is often used extensively in both developing and displaying creative works, the lack of a customizable user interface can significantly influence the practical experiences. In other words, being able to adjust the visual appearance according to user preferences could contribute to increased productivity through making the patch more pleasant and clear to view.

More specifically, the customization refers to the ability to adjust the color settings of all the elements within the patch. This includes canvas background, object background and foreground, connection wires and elements such as object box and inlet/outlet. The customization should also allow different fonts and their related properties to be used in the patch. The settings of personalized parameters can be accessed either through an intuitive graphical editor, or by editing a plain text configuration file.

Furthermore, additional elements may be introduced into the graphical user in-

terface to increase usability. For instance, a button bar can be integrated into the canvas window for the object to be quickly created by clicking on the appropriate icon[3]. This type of design can also be used for implementing features such as object search and displaying various status information. Furthermore, users should also be able to remove unnecessary visual elements if required. Menu bar, scroll bars and any additional components can therefore be switched off so they will not be displayed.

In short, through adjusting the appearance of the graphical user interface, users can work more comfortably according their needs and preferences. This is contrary to the current design of Pure Data, where users are constrained by very limited options in its visual customization. Through such implementation, The flexibility of Pd as a programming environment also reflects how its interface may be customized. Furthermore, this also allows a more polished user interface to emerge, which adds additional advantages to the program. For instance, one of the main arguments in the comparison between Pure Data and Max/Msp is often focused on differences in the level of sophistication of the user interface. Having such improvements would certainly regain Pd's competitive edge in this regard.

4.3.2 Lack of optimized command invocation

After using Pure Data extensively, one can quickly recognize that certain operations and commands are generally invoked more often than others. This is especially true in the case of patch editing. In current Pd, many of these frequently used commands have suboptimal method of invocation. In other words, performing these operations are generally not intuitive, or involve a fastidious process. They thus become problematic in the efficiency of Pd programming, as additional time and attention is needed to complete these tasks. Furthermore, it also steepens the learning curve for new users to become accustomed to Pd's operations.

To demonstrate the inconvenience, a few examples will now be provided and

[3]Such button bar already exists in Max/Msp

discussed. A new object is typically created via the menu selection or the "Ctrl+1" key shortcut. While object creation is a simple process, instantiating the new object after typing in its name and arguments is somehow less obvious. Intuitively, one would expect pressing keys such as the "Enter" to complete the creation process. However, it instead requires moving the mouse pointer outside the the object's bounding box and then followed by left click once on any blank area of the canvas. Although the "Enter" key arguably may not be the most suitable choice, if multi-line objects are allowed, it nevertheless highlights that the process of object completion could be shortened with the use of key bindings. Therefore, one of the drawbacks in current Pd is the additional mouse movement and awkward object completion. This is particularly cumbersome when object editing takes place very frequently.

As Pd relies on graphically interconnecting objects to form the final patch, users thus regularly have to attach and detach wires between objects. At present, the mechanisms to aid the operations on wires are very inefficient. For instance, wires can only be created by moving the mouse pointer to the target outlet, then clicking and holding the mouse button, to be released once the pointer has moved to the desired inlet. This means that only one wire can be established at a given operation. More over, once a wire is made, the only method to modify the connected object path is to first delete the wire, then reconnect the necessary objects. It is foreseeable that better methods may be conceived so that wires may be created and edited more rapidly, without repetitive mouse movements and clicks.[4]

Not having the support for multiple levels of "undo" and "redo" has clear disadvantages in patch editing. As Pd traditionally only provides immediate "undo", tracking and recovering from unwanted edits several steps back is therefore a troublesome task. The implication of such a feature can be extended beyond error recovery. This is because it essentially saves and retrieves each editing step made by the user, and therefore, it is possible to both export and import his-

[4]Methods of evaluating the efficiency of different user interface designs can be found[30], which could be employed to further investigate on improving Pd usability. In particular, the GOMS keystroke-level model originally proposed by Card, Moran and Newell[6] could provide a comparative test between the use of the mouse and the keyboard in controlling Pd.

torical data recorded. This would allow patches to be scripted and constructed automatically.

Lastly, since Pd consists of many external libraries, the number of available objects can be very large. To utilize all the objects, users often have to read their corresponding help files so they can be used correctly. Pd conventionally offers a rather limited method in which help files can be accessed. A simple key shortcut to load the appropriate help file would shorten the process of selecting from a pop-up menu activated by the right click over selected object. In addition, users should also be able to browse all available objects and load its help file without having to instantiate the object first.

Implementations of this kind, as a result, aim to improve the existing functionalities for Pure Data to be more practical and convenient to use. They will hopefully not only enhance the experiences of programming for existing users, but also make it more accessible to artists who wish to learn Pd for the first time.

4.3.3 Lack of utility features

In this category, utility features refer to additional tools which can help users in programming or using Pd. These features may not directly increase the performance of the program, but will nevertheless allow users to explore the full advantage of Pd as a creative programming environment.

As mentioned previously, it is not uncommon to have a considerable number of available objects in a particularly Pd installation. Memorizing the names of objects thus becomes a difficult task, especially for objects not frequently used. For this reason, an auto completion feature would be useful for helping users to create the necessary object. Moreover, it can increase the speed in object instantiation, without having to type completely its exact name. Eliminating misspelling would also be an additional benefit. This feature can be further extended to auto complete the creation arguments of a given object, which clarifies the required parameters.

A feature that enables zooming would allow the display of a patch to be quickly adjusted. This would be particularly useful in the context where an overview of a large patch is required, or the attention is needed to focus on a specific area for the detail. Despite the cosmetic nature of this feature, it can have some practical effects on the way Pd patches are presented. Being able to change display size meant that a patch can be conveniently shown to obtain the best result.

Traditionally, the placement of objects on canvas is arbitrarily chosen by the users. While this gives users the ability to program intuitively, the visual structure of a patch, however, is often compromised. In other words, as the complexity of a patch evolves, it becomes increasingly difficult to read and understand it. This is especially problematic when continuing a previously interrupted programming project, or when the development involves collaborating with others. It is therefore conceivable to have a feature that rearranges the patch according to a certain hierarchical order, so that its functioning and purpose can be easily understood. This could allow users to visualize their patch from a different perspective, thus encouraging creativity.

Patch encapsulation could offer effective mechanisms to increase the speed in developing extensive patches. It means that users can efficiently group parts of the patch and automatically pack them into sub-units, so that the patch overall becomes more modular and easier to maintain. Encapsulation could possibly be further developed to support abstractions. This would then enable packed content to be reused from a different patch. This feature, to a certain degree, would remove the task of patch organization away from users' responsibility, so they may focus better on the development of their works.

Since the criticisms so far have been based on improving the existing functionalities of the graphical user interface, or proposing additional features to it, the key issue thus lies in the usability of the program. In other words, these modifications do not have direct impact on Pd's internal structure or optimization. They would however allow Pd to be used more efficiently and effectively from the users perspective. Although many of its internal designs can also be further developed, its usability is arguably by far the weakest aspect of Pure Data.

Furthermore, as many extremely complex patches have been produced in the past, it meant that it is already capable of supporting extensive programming, despite some of its internal shortfalls. The implementations of its graphical user interface, on the other hand, has long been a subject of major critique that demands significant changes.

4.3.4 Client and server architecture

One potential improvement on the internal design of Pd can be found in the separation of the client and the server. The client refers to the part of the program that the user interacts with to create their work, whilst the server performs the necessary computation to fulfill the tasks that the user has programmed. Through the client, patches made by the user are sent to the server, so that it can be executed. The graphical user interface, for instance, can be seen as the client. Components such as the DSP and the dataflow engine therefore belongs to the server. This client and server architecture has been a common practice in software programming to allow the resulted application to be more flexible. In the context of sound synthesis programming environment, SuperCollider is perhaps one of the most notable examples in adopting this method.

Although Pd already consists of a client and a server, the separation between them is not fully implemented. This is evident by the fact that the user input is entirely interpreted by the server. In other words, apart from the core functionalities in executing a patch, the server is also in charge of computing how it is rendered and edited in the user interface. The Pd client, essentially does not perform any computation, and essentially consisting a collection of functions which are called by the server. As a result, the tasks clearly are not assigned effectively between the client and the server, and can have significant effect on the program's performance overall.

For Pd to achieve the client and server separation, tasks that are not part of core functionalities must firstly be removed from the existing server. They may then be reimplemented in the client. The server, therefore, only has to provide the necessary critical operations. The client will be assigned to determine how

a patch is visualized and notify the server when changes are made by the user. A protocol would be required to allow the server and the client to communicate. The communication should ultimately be optimized so the bandwidth needed is kept minimum. The protocol could be based on existing standards, to encourage furtherer innovation in future developments.

Such a implementation would allow the internal operations of Pd to be restructured, which can bring several advantages. First, it would significantly clarify the existing code base of Pd into distinctive components. This would enable developers to easily focus on the necessary part of Pd. This can have significant effect on the long term maintenance and improvements of the program, as each independent part becomes ever more optimized. The modular approach would also give greater flexibility in the use of Pd. Different types of client can be created to suit particular purposes. The server, on other hand, can be accessed over the network. Furthermore, it is foreseeable to have multiple clients connecting to the same server or vice versa.

4.4 Branches of Pure Data

Having identified several possible areas on which the practical work may be based, the investigation then moves towards focusing on how these implementations should be carried out within the existing Pd community. A closer examination of Pd's model of development would also be necessary, to successfully apply the proposed works.

Currently, Pure Data has a on-line repository[5] containing all of its source code. Through the use of version control software[6], developers can collectively modify these source files, and coordinate the efforts between them. A mailing list exists to aid the discussions on development related subjects. This method of collaboration is a common practice in FLOSS projects. Over the past years[7], two

[5]Pd's source is is hosted on SourceForge, one of the largest on-line repository for FLOSS projects. http://sourceforge.net/

[6]Pd community uses a software called CVS (http://www.nongnu.org/cvs/) to coordinate development efforts

[7]Since Pd version 0.35

dominant branches of source code have emerged within the developer's community. First, the "main" branch, which consists of codes originally released by the project leader, Miller Puckette. Consequently, he is also the sole maintainer of this branch. Secondly, there is a branch called the "devel", which developers collectively work on. One of the leading developer, Tim Blechmann, has been the maintainer of this branch.

The main branch, as its name implies, is considered to be the stable version of Pure Data, and for this reason, it is often regarded as the official release. The source code of this branch has mostly been developed by Miller Puckette alone. For others to contribute, patches must first be submitted to the maintainer and then await approval for inclusion. In other words, the main branch has a very centralized development model, moderated by the project leader.

The "devel" branch, on the other hand, is built on equal collaborations between developers, who mutually dictate any work in progress. For this reason, this branch is considered to be the experimental version of Pure Data. Developers are free to propose and experiment with features which do not yet exist in the main branch. Although the development process of the devel branch is far more distributed, major changes still need to gain group consensus prior to a final commitment. Due to its experimental nature, some implementations in the "devel" branch have allowed it to be more optimized than the main branch.

Having these two branches means that significant modifications can be initially tested and evaluated in the "devel" branch, and once prove successful, may then be eventually included in the stable release. The original aim was to synchronize these branches periodically[8] so their differences could be merged. This would allow the two branches to mutually benefit from their individual progress. As the devel branch follows the changes made by Miller, it keeps the branch up to date with the current release. Similarly, by incorporating new features from the devel branch, the main release can therefore extend its functionalities. Having a stable and a experimental branch of the same software is often observed in FLOSS project, however, what is uncommon in the case of Pd is that the project leader is not involved in developing the experimental branch.

[8]The initial aim was to merge between the branches every 4 months

In 2004, a temporary branch was created and named "Impd". It was as a proof of concept project proposing many radical and fundamental changes. One of the underlying foci of Impd was to ultimately achieve the client and server separation for Pd. It was maintained and initiated by Mathieu Bouchard, an active developer in the community. He is also the author of "GridFlow", a multi-dimensional array processing library for Pd. Although Impd clearly demonstrated how such modifications could be implemented, it ceased being developed shortly after its presentation at the first Pure Data convention that year[9].

Several conclusions became apparent after gaining some knowledge relating to Pd's development model. Since the practical project attempts to implement some rudimentary changes to existing Pd, it is not appropriate as part of the main branch. The devel branch thus becomes a natural candidate for this project to be based on. Furthermore, many of the concepts found in Impd are similar to what the practical project proposes, and could potentially offer a good starting point for the project. In addition, to complete all the proposed improvements will be a extensive undertaking as an individual, due to personal ability and limited time. Seeking collaboration and advice from existing Pd developers would therefore be crucial in achieving its goals.

Having contacted the author of Impd, discussions on various issues surrounding both Impd and the proposed project, quickly took place. Impd was no longer in development due to the lack of initial interest from the developer community. Although the reasons behind such a response were not fully understood at this stage, they later became clear and consequently influenced the project. In spite of this, Mathieu still remains deeply interested in several concepts in Impd. After some correspondence, the possibility of collaboration materialized, thereby forming the basis of the practical work in this research.

[9]According to Mathieu Bouchard, during the first Pure Data convention, Impd did not gather enough interest and response for him to carry on developing the project alone.

4.5 DesireData

In late 2005, the collaboration was formalized in a new project called "DesireData", aiming to carry on the initiative set by "Impd". The main objective of the project is firstly to achieve client and server separation, and secondly to implement various usability features mentioned previously. In addition, it will remain as compatible as possible to the current Pd, including its file format and external libraries. This would allow existing users to easily test DesireData, and also enable the project to inherit the extended functionalities in Pd.

A few options were available as to how DesireData should be developed. It could follow the previous example of Impd, where it would be a separated branch of its own. Alternatively, the project could be part of the devel branch. For practical reasons, it was decided that DesireData should adopt the latter approach instead of the former: for two developers, DesireData would still be a significant undertaking. It was hoped that by being part of a recognized and active branch, it would be possible to attract other developers to take part in the project.

However, because DesireData requires fundamental changes to be made on Pd's source code, its integration into a working branch would have to be carefully planned. This ensured that it would not introduce additional maintenance for developers in the devel branch, whilst allowing the project to evolve freely. In other words, a practical solution was necessary for the devel version and DesireData to efficiently coexist in the same branch. As a result, the initial phase of the project took shape as a compile time option, where all source files related to DesireData are only included if such an option is activated during compilation. By doing so, the modifications and source codes of DesireData can be kept separated, thus solving the issues of integration.

The practical element of this research is now apparent and can be clearly outlined. By participating in DesireData, it will aim to address various problematic areas identified in the previous section. Working closely with other developers would contribute to a greater insight into the subject. It will also work towards achieving the original goals of Impd. Mathieu being a highly skilled software programmer and with extensive experience working with Pure Data, the col-

laboration would greatly benefit the practical work from a supervision point of view.

The main goal of Desiredata, therefore, is to enhance the everyday experience of Pure Data from the perspectives of both user and developer. A more effective and user-friendly interface should be expected for normal users. It will be more extensive for customization and contain features which would help users to program more efficiently. The change of internal architecture would mean that developers can quickly extend its functionality. It will also reduce the overall size of source codes, making it more accessible and easily maintained for the future. The practical work will thus be mostly focused on the implementation of the improvements in the graphical user interface, whilst Mathieu will take charge of the internal design and modifications.

4.5.1 Methodology

The methodology employed in DesireData consists of several major steps, listed below:

- Server modification
- Client infrastructure
- Client reimplementation
- Additional features

In modifying the server, the focus will begin with removing any unnecessary elements. These include parts involved with the handling of user input and rendering of graphical elements. The server will no longer compute any information regarding the visual aspects of a patch. Objects, will be drawn entirely by the client, without the server knowing their representation. Additionally, other mechanisms need to be implemented to allow the client connecting to the server. The main task of these mechanisms is to provide synchronization between the server and the client.

Client infrastructure refers to the fundamental designs that later implementations will be based on. One of the design principles is to avoid repetitive coding and encourage code reuse, by means of adopting object oriented programming. For this reason, components in the client can be better organized, without duplicating previous works. This will also enable future developers to rapidly add new elements. As a result, a customized object oriented system, named poe.tcl, was created by Mathieu to suit the purpose of DesireData.

Once the infrastructure is in place, the next step is to re-implement the functionalities of the existing graphical user interface, using the methods provided by poe.tcl. The aim of this stage is to produce a functional user interface which replicates most of the aspects of the conventional Pd. This is key for providing the essential familiarity and compatibility for end users. Furthermore, various fundamental infrastructures, such as poe.tk, can be extensively tested and further refined during the reimplementation.

Lastly, additional features may be introduced into DesireData, fulfilling its original objective - these include new features for both the server and the client. However, in the context of this research, the main concern lies mostly in the improvements which can be achieved for the client. Many of the features will be derived from the limitations of existing Pd, identified previously.

It is worth noting that these different steps do not necessarily take place sequentially. In reality, they are concurrent tasks which are constantly being revisited. However, they do offer a methodological approach in reviewing the overall development of the project.

4.5.2 Design principles

Having established the project methodology, the design patterns involved in DesireData will now be presented. The main objective is to adopt techniques that would optimize the development process. This includes being able to efficiently achieve the goals set by the project, and to minimize efforts in long term maintenance. Most of these principles aim to bring together a better organization of code, by reducing its size and to modularize the existing program into distinct

components. In short, these principles will hopefully provide an effective and long lasting foundation for the project.

Avoid code duplication

This notion is critical to improve productivity and efficiency in developing DesireData. The aim is to examining the overall characteristics of the program and merge similar tasks into generic parts. As a result, the program will eventually be highly modularized. This not only unifies the source code into distinctive components, but also reduces its size by a significant amount. Furthermore, this will benefit the readability of the source files, thus making them more accessible and easily maintained. This concept is often described by using the term OnceAndOnlyOnce (OAOO), originated from the Small talk and Extreme-Programming communities. DRY(DontRepeatYourself) is another terminology used in the Ruby and Pragmatic Programmers communities to categorize this approach. The advantages of code reduction can be summarized in the following two points

- Merges bugs together so that there are less of them to fix. The best debugging technique is to write less code.
- Merges features together so that there are less of them to update. By centralizing behaviors relative to families of features, new features may be implemented easily. Think of coding a new feature that has to interact with ten existing features. Now imagine coding a new feature interacting with only one feature which is the common part between the ten features just mentioned.

The merge of two radio-button classes, horizontal and vertical, demonstrates the effectiveness of this principle. The code specific to radio-buttons is measured about 53kB in conventional Pd. In Impd, it was reduced to 9kB. DesireData, on the other hand, only consists of 181 lines of code that are unique to radio-buttons. Tasks such as handling and rendering radio-buttons in the correct orientation are abstracted into generic methods, which are shared with other

graphical elements such as the slider class. Furthermore, any future implementation on orientation sensitive GUI elements can quickly make use of these methods.

Object-Oriented Programming

One effective approach to achieve better organization of code lies in the domain of Object-Oriented Programming(OOP). Because common elements within the programs are abstracted into generic classes with inheritance relationships, this makes the resulting program very modular and reusable. In conventional Pd, OOP is already implemented in the server using C. In order to efficiently decouple the client from the server, it is obvious that the client would require its own OOP system.

Poe.tcl was consequently created to facilitate Object-Oriented programming in the client. It provides the OOP mechanism written entirely in Tcl, the language of choice in Pd's graphical user interface. As Tcl is not an Object-Oriented language, an external system was therefore necessary to enable OOP. Although a few other systems aiming to enable OOP in Tcl also exists, none felt suitable for the purpose of DesireData. One of the main reasons was that by designing a customized OOP system, the syntax which will be employed when building the client can be very flexible. This would ultimately enable a smooth migration in the future if the client is to be ported to a different language. Moreover, having poe.tcl also meant that the distribution of the client can be made very easy, without increasing its dependencies on external software packages. This is because not all existing OOP extensions of Tcl are widely accessible. Furthermore, since Tcl can be extended with additional semantics with a remarkably little amount of Tcl code[10], maintaining and distributing such a minimal customized OOP system is highly practical.

Following the conventions in OOP, classes in DesireData client define a set of generic abstract entities. Most of the classes are created according to a hierarchical order, in which the inheritances on methods and variables are based.

[10] poe.tcl currently consist of 276 lines of Tcl code.

To initialize a class, the syntax *class_new foo* {*bar1 bar2*} is used, where *foo* is the class name. *Foo* also declares its super classes to be *bar1* and *bar2*. Note that multiple superclasses are permitted. The *class_new* method further defines *CLASSNAME_new_as* and *CLASSNAME_new* procedures to allow instances of a class to be constructed using a specific ID or a generated one. Once a class is created, the syntax of *def CLASSNAME* {*arglist*} {*body of method*} can then be used to define class method.

Once objects are instantiated, the syntax used to call their methods and send messages to objects is similar to most other OOP systems. The convention is often described as the *subject-verb-complements* syntax, or also known as *receiver-selector-arguments* syntax. The first part refers to the unique name or ID of the object, and the second is the name of the method to be called. Any additional elements after the first two are treated as arguments to the method. For example, *canvas editmode= 1* will call the method named *editmode=* with the argument of *1* on the object called exactly *canvas*.

The underlying mechanisms in poe.tcl to enable method definition and calling is achieved by the combination of *proc def* {*class selector arglist body*} and *proc lookup_method*. For instance, by first calling *def objectbox draw* will actually define a proc named *objectbox_draw*. Then, *proc lookup_method* will reconstruct the appropriate proc name from any *receiver-selector-arguments* calls and invoke the corresponding proc. To give a realistic example, the message *"825ea50 init 450 500 +0+0 1"* will initializes object *825ea50* with the arguments *450 500 +0+0 1* and the method called is *canvas_init* with *825ea50* inserted at the beginning of the argument list. In this case, *825ea50* is the ID of an object of class *canvas* or any subclass of it which does not redefine *init*.

In poe.tcl, the "@" sign is the prefix which denotes instance variables. This notation is a significant improvement from Impd, in which uses variable conventions of Tcl. Instead of typing *_($self:x)*, all its needed now is *@x* instead. Furthermore, this enables trivial modifications on the existing code to take advantage of the new *dict* feature in Tcl 8.5.

Model-View Separation

In existing Pd, because graphical rendering is mostly handled in the server, there is essentially no difference between an object and its visual representation. The Model-View design, therefore, aims to further abstract all GUI objects into two parts. It consists of a generic element (Model) that stores the current state of a object, and a rendering part (View) that is in charge of computing the graphics of an object. This separation is necessary to achieve the client and server architecture. By having the View implemented in the client, the server no longer requires to take care of the patch visualization. Furthermore, as long as the means of synchronization is available between the Model and the the View, the client can be used to control the server remotely. Another advantage of this design is that the graphical representation of objects may be easily changed, it is also conceivable to allow multiple Views to be connected to the same Model.

The Model-View implementation is similar to the well known Model-View-Controller design in software engineering. In DesireData, the Controller and the View are the same entity. This is due to the fact that there isn't a need to implement them individually in the present context. Additionally, the Model exists in both the server and the client. This minimizes the potential traffic in communication by caching the Model's data in the client.

Observer-Observable

An Observer-Observable pattern is used to handle the communication between the Model and the View in DesireData. Each object is observable, and can be subscribed to an owner. The owner is thus capable of noticing any changes requiring updates within all the objects it observes. For example, a text object is subscribed to the canvas it belongs to, and its changes will then be noticed by the canvas. Through this relationship, all notices are propagated towards the root of the hierarchy and managed centrally. This enables updates to be prioritized by removing any duplications and delaying frequent changes. In fact, there are two Observer-Observable trees per patcher window in Desiredata. One organizes the models and sends updates to the views on the client, the other

manages the views' update to the Tk canvas. Naturally, the first tree resides in the server, and the second exists in the client. The scheduling of these two update trees are governed by using *t_clock* in the server and the *after* command in the client. Both mechanisms are functionally equivalent - the former being a feature of Pd and the latter is a Tcl command.

4.6 Social context

Alongside technical details, the sociological aspect of DesireData is also the focus of this research project. As the software development model in FLOSS promotes distributed collaboration, taking part in the development process would provide an opportunity to closely examining such a phenomenon, thus gaining a greater insight and experience. This section, therefore, aims to provide the social context of DesireData, and outline the key issues of the project. It will start by presenting the community responses to the project, then highlight the problems encountered, and finally document the findings and conclusions.

Shortly after the initial announcement of DesireData, it began receiving feedback within the developers' community. While many welcomed the new initiative, others opposed it. Although these responses were anticipated to a degree, they nevertheless pointed out a few interesting characteristics of the community. The opposition of opinions were distinctively clear, giving an inharmonious impression. In other words, despite differences of opinions and individual objectives, constructive discussions on the subject were very limited. The project leader, however, did not express his views on the project nor did he comment on the resulting responses[11].

The objection argued that features planned for DesireData were too radical, and focused too much on the change to user interface, instead of further improving the core functionalities. Although DesireData indeed proposes several

[11]This is clearly evident by the various communications that took place on the pure data developers' mailing list, where Miller had not responded to the mixed opinions on the subject. The archives of the above mentioned mailing list can be found at http://lists.puredata.info/pipermail/pd-dev/

fundamental improvements that are arguably radical, this is precisely the reason it has chosen to exist in the devel branch. Hence, concepts in DesireData can be implemented and evaluated without affecting the main line code base. Furthermore, changes in DesireData are far from being superficial, since much of its design aims to improve the internal structure of Pd.

Reactions from the users community was clearly in favor of DesireData's initiative. Many of them agreed with the problematic areas the project aims to resolve, in particular the proposed improvements to the graphical user interface. These responses were highly encouraging, as the project successfully identified the demands from the users' perspective.

As the project progresses, it becomes clear that there are two opposing groups involved within the developers' community. On the one hand, some developers follow the programming approach and the pace set by the original author. On the other hand, dissatisfied with the design and the rate of change, the other group of developers are eager to take initiatives and experiment with new improvements.

Within the first type of developers, they often have a more conservative approach in the community. In other words, although they are actively involved in the development, the decision making process is often centralized. Important issues may be objectively discussed prior to the change, but the acceptance by the project leader is still critical in gaining the consensus and support amongst the group. For this reason, they typically value qualities such as the coherence, the stability and incremental changes in software. This is because modifications are made by a few people only, and the rate of progression is less then rapid. For example, in the main release of Pd, Miller has the final authority over the features included for the coming release, and he is the only person who decides the frequency of the release, which typically is once a year.

In general, the second kind of developers are more critical towards the design and implementation of Pd. This by no means disregards the past achievements of the program, however, they are constantly looking for new potential improvements to take Pd one step further. For this reason, they often take initiatives

attempting to engage the community to innovate. For instance, they organized monthly on-line developers' meeting in the past, the aim of which being the establishment of more direct communication between developers thus to better organize various internal tasks. From their perspective, software development should be an open and decentralized collaboration. In other words, not only does everyone have equal opportunities to raise their concerns, but can also decide on important matters. The decision making process, therefore should be objectively made by the majority. Lastly, they highly appreciate the value of innovations and are clearly motivated to carry out experiments and implementations.

Developers having contrasting views and characteristics often exist within FLOSS projects. Such a phenomenon arguably contributes to the rich social dynamic within the community thus further motivating collaboration. However, the dissonance observed amongst Pd developers is less than ideal and was not anticipated. Since it is not uncommon that FLOSS consists of developers with different opinions, further investigation is needed to identify the critical factor that leads Pd into the current state of affair.

One aspect that appears to be unique in Pd is the role played by the project leader. Although Miller has been known for maintaining the main line code base and not the devel branch, he has rarely voiced his concerns over numerous disagreements within the community, including issues with DesireData. This could be clarified by briefly examining the history of Pd as a collective FLOSS project.

Prior to version 0.35[12], Pd was already released as a FLOSS project under the BSD license. At this time, because Miller was the sole developer of the project, Pd had no official collaboration platform such as on-line source code repository and version control system. As Pd became evermore popular, it began to gather an increasing number of users and developers. There was now a need to coordinate between the individuals involved. As a result, initiatives were taken by the early adapters to establish various development facilities in an attempt to formalize the emerging collective. Members of the community guided

[12]Including version 0.32, 0.33 and 0.34

Miller through the transition so that, as the project's original author, he could efficiently utilize the available tools. However, although Miller had adapted to the newly formed collaboration, the previous development method nevertheless remained similar to a certain degree. This is most evident with Miller's annual release, where most of the changes made in the devel branch over the years are not included. In other words, despite the growing community, Miller maintains a level of isolation in his approach to the development of Pd. This condition holds true until present day.

The role of project leader in FLOSS is far from trivial, despite its distributed and decentralized appearance. Project leaders need to have the ability to not only moderate but also consolidate the community resource. This can be particularly troublesome when the development process is transparent and each of the individuals involved hold their own perspective on the subject. While there is no convention in the approach to project management, some common trends may nevertheless be outlined.

In many successfully FLOSS collaborations, a typical trait of a project leader is the ability to clearly articulate and formulate his or her own views to the community. For this reason, the main approach and original objectives of the project are precisely defined without confusion. Furthermore, he or she has to be able to mediate between developers of different opinions, so a group consensus may be maintained. These characteristics can be found in the cases of Richard Stallman, Linus Torvald and countless other prolific FLOSS projects. Richard Stallman leads the GNU project with his uncompromisable vision, while Linus Torvald's confidence and diplomacy allows Linux to achieve its goals.

Miller's isolation, therefore, has resulted in the lack of a visible central figure within the community. This is particularly problematic when disagreements occur. Without mediation, they may quickly escalate into intense debates which generally lead to a negative outcome. As the foundation of a project is generally set by its original author, his or her opinions in such a scenario often provides an effective means of clarifying issues in the dispute. Furthermore, without an authoritative intervention, a community may easily lose its focus and the collaborative effort may thus be dissolved. Evidence of divergence in development

can already be observed. Lastly, the isolated approach can also influence the overall outlook of the community. By not actively engaging with the community, developers are thus less likely to collaborate under common interests. Individuals tend to act according to their own personal agenda, rather than addressing fundamental issues collectively.

Since it was not Miller's original intention to formally establish Pd with an open collaborative environment and subsequently be recognized as the project leader, there is no obligation for his engagement with the existing community. For this reason, Miller is by no means entirely responsible for the development of Pd, as a FLOSS project. In other words, Miller's choice of FLOSS license for Pd and the consequent activities generated as a result are separate incidents. It nevertheless highlights the importance of leadership in FLOSS development and the extent to which a project leader can influence a community.

Based on the observation above, one may conclude that Pure Data, as a software project, does not reflect particularly well on the development model found in FLOSS. To briefly summarize the reasons, firstly, it was the early adapters of the software who formally established its FLOSS community, not the original author. As a result, the position/status of the project leader became ambiguous. According to the conventions of FLOSS, Miller would be regarded as the project leader, and this is still the general consensus within its community. On the other hand, as Miller did not initiate the community, he therefore is not formally obligated to act as the project leader. For this reason, his absence from the community's activity can thus be understood.

The internal divide, with the lack of mediation, has recently produced some significant changes. Developers who represent the second and more radical group have begun to show signs of losing interest in the involvement of Pd community. For instance, the devel branch, which is traditionally maintained by these developers, is no longer in operation. In other words, there is no corresponding experimental branch for the latest version of Pd's main release. Most importantly, these developers have diverted their interests to new and similar projects which arguably derived from the frustration experienced within the Pd development.

Tim Blechmann, who has been a long term maintainer of the devel branch, officially announced his departure in late 2004. He is now focusing on developing his own audio dataflow programming environment, which aims to completely redesign many of the fundamental shortfalls which he had identified in Pd. As a result, it aims to bring highly optimized performance to audio synthesis, and ultimately multi-media programming. It currently has a preliminary testing version available and is named Nova. While no longer actively maintaining and developing Pd, Tim continues to stay in contact with the Pd community through the mailing list.

Thomas Grill, another key figure in the devel branch, has also started an independent project based on Pd. Named Vibrez, it aims to modify the existing Pd to offer a unified and comprehensive software package that perform consistently over a variety of operating systems. It will eventually have its own graphical user interface. Not being a FLOSS project, many details on this project are not disclosed to the public.

Discouraged by the lack of communication within the community, Mathieu Bouchard resigned from Pd's mailing lists in 2006. While not formally engaging with other developers, he continues to moderate the official Pure Data IRC channel. Furthermore, Mathieu is currently focusing on the development of DesireData, which still resides in Pure Data's on-line source code repository.

Since the devel branch is no longer being developed, DesireData is now a independent branch of its own. This decision has brought some practical benefits to the project. It has allowed DesireData to be developed freely, without interfering with the works of other developers.

4.7 Current Status and Conclusion

Although the development of DesireData was subject to several interruptions, as of September 2007, the majority of improvements it originally set out to implement were completed. Each of the eight pre-releases since September 2006 have shown a steady progress towards completing the goals of the project. These

releases also gave other developers and users opportunities to test it and provide valuable feedback on different aspects of the project. Although Mathieu Bouchard and Chun Lee still remain the two main contributors to the project, others have shown interest in being involved. As a result, other projects have been initiated with the help of these early adopters, to support the continuation of DesireData.

At present, DesireData has a web portal which consists of a ticketing system and a wiki[13]. The purpose of the former is to better assign and monitor the development process, whereas the latter is intended to facilitate the documentation of the project. Alongside this, a dedicated mailing-list also exists for individuals to discuss current issues and bug fixes. Lastly, a large part of the communication also takes place over the project's IRC channel. Such a method of correspondence is much more rapid and thus more effective than the list. As a young and developing software project, DesireData currently has fifty-eight subscribers on its mailing-list and around ten members (On average) on its IRC channel.

Two supporting projects were derived from DesireData: "PureUnity" and "Patching-in-tongues". The goal of PureUnity is to develop an automated testing suite for DesireData. As the complexity of a given software increases, testing becomes evermore important in the process of development. By conducting automated tests against new modifications in the code base, it would allow potential bugs to be caught early and systematically. Furthermore, based on the principles of unit testing, any previous bugs and new features have corresponding test cases in PureUnity, thus ensuring the long term quality of the software. In other words, having a extensive testing suite can lead to less unknown bugs in the source code.

"Patching-in-tongues", on the other hand, aims to internationalize DesireData by providing locale support. This means elements such as menu items, pop-up dialogs, tool-tips and object descriptions can have a multi-lingual display so that users who are less familiar with the English language can operate the program. At present there are twelve languages being translated by a team of seventeen

[13] http://trac.edgewall.org/

contributors. Moreover, because of the Unicode support, locales included are not limited to languages based on Latin characters. For instance, Japanese and Chinese are also supported in DesireData. The ability to internationalize Pure Data has been a long term discussion within the community, and being able to achieve this at this stage in DesireData is yet another significant improvement over both its traditional and proprietary counterparts.

To date, DesireData has been presented in several festivals and conferences. These are crucial opportunities for the project to gain more interest, as well as receiving critical feedback. Furthermore, as DesireData developers are located in different parts of the world, these events also provided rare chances for them to meet in real life. Some of the key events will now be given as examples.

DesireData took part in the Piksel festival in both 2005 and 2006. At the first event, it was accepted as a paper submission, where its design and theory were described. This paper also marked the official public announcement of the project. As a result, DesireData was once again invited the following year to demonstrate the software and present the subsequent progress. The positive feedback and interest received at the Piksel festival proved to be a real encouragement to the project in its early development.

The live performance by the Canadian artist Robert Atwood[14] at the live coding conference at Sheffield, England in 2007, is perhaps the first real production made using DesireData. In preparation for the performance, Robert worked closely with both Mathieu Bouchard and Chun Lee to test the software and give design suggestions. As his performance focused on the practice of live coding in the domain of visual dataflow programming language, several user interface features in DesireData proved to be particularly useful in such a context.

Most recently, DesireData was presented at the second Pure Data convention in Montreal in August 2007. For many Pd developers, this was the first time they saw the project and were able to directly discuss many of its ideas. In particular, the improved keyboard based program control and the automated testing methodology formed two distinctive subjects in the paper sessions. Throughout

[14] http://robert.lurk.org/

the convention, many members of the community showed increasing interest and acknowledgment of the extensive works carried out so far in the project.

Although most of the features in DesireData are already implemented and can be demonstrated, it still falls short when employed in large scale productions. The main reason for this limitation is due to the sub-optimal performance of the widget toolkit inherited from the original program. In Pure Data, Tk was used as the toolkit to render the patcher window. As both the responsibility and the functionalities of Pd's GUI are strictly limited, the effect of Tk's performance was less obvious. DesireData, on the other hand, has its client written entirely using Tcl/Tk, and is managing all user related tasks together with additional features, the effects of Tk X have thus become apparent and extensive. Despite the performance of Tk having always been known as less than adequate, the full extent of its effect on DesireData was unfortunately discovered rather recently. As a result, this is currently the only remaining factor preventing the project from its first general release.

Several potential solutions have been found and are currently being investigated. First, it is possible to directly modify the source code of Tk widgets and early experiments have shown promising results. However, this can bring additional complications when distributing the software. This means DesireData will either maintain its own branch of Tcl/Tk, or depend on the testing version of Tk, assuming its modifications will be accepted by Tcl/Tk developers. Each of the options remains to be further evaluated. Secondly, it is possible to switch the widget bindings of Tcl to other toolkits. For instance, Tkzinc[15] and Gnocl[16] offers alternative bindings using OpenGL[17] and GTK+[18]. Because of the modular design in DesireData, changing widget toolkits can be made rather efficiently. However, due to the lack of benchmark documentation amongst available alternatives, making the right choice is less than trivial. Having found the possible solutions in the above two categories, it was decided that they should be explored in parallel, so as to minimize the risk of repeating the experience with

[15] http://www.tkzinc.org/tkzinc/index.php
[16] http://www.dr-baum.net/gnocl/
[17] http://www.opengl.org/
[18] http://www.gtk.org/

Tk.

The experience of working on DesireData also highlighted several areas of interest which could lead to further research. Since DesireData allows users to interact with the program in several different ways, it would be interesting to conduct comparative tests on the efficiency between different input strategies, as well as statistically recording common user behaviors for quantitative analysis. It is conceivable to devise multiple representations of the same dataflow patch. In other words, patches can be visualized according to different properties, thus enabling users to better explore the inner structures in them. Lastly, automated graphing and grouping could significantly reduce the amount of human effort spent on visually managing objects and patches. In short, not only has the original objective of the project produced positive results, it has also revealed possible directions for future developments, for both DesireData and personal practice.

Selected Bibliography

[1] Bentley, P., and Corne, D. W. *Creative Evolutionary Systems*. Morgan Kaufmann, London, 2002.

[2] Bezroukov, N. Open source software development as a special type of academic research (critique of vulgar raymondism). *First Monday 4*, 10 (1999).

[3] Bezroukov, N. A second look at the cathedral and the bazaar. *First Monday 4*, 12 (1999).

[4] Boden, M. *The Creative Mind*. Routledge, London, 1990.

[5] Bohm, D. *On Creativity*. Routledge, London, 1998.

[6] Card, S., Moran, T., and Newell, A. *The Psychology of Human-computer Interaction*. Erlbaum, Hillsdale, N.J., 1983.

[7] Collins, N. Generative music and laptop performance. *Contemporary Music Review 22* (2003), 67–79(13).

[8] Cope, D. *Experiments in musical intelligence*. A-R Editions, Inc., Middleton, 1996.

[9] Coveney, P., and Highfield, R. *Frontiers of Complexity*. faber & faber, New York, 1995.

[10] Dawkins, R. *The Blind Watchmaker*. Penguin Books, London, 1986.

[11] Dostàl, M. Genetic algorithms as a model of musical creativity - on generating of a human-like rhythmic accompaniment. *Computers and Artificial Intelligence 24*, 3 (2005).

[12] Draves, S. The electric sheep screen-saver: A case study in aesthetic evolution. In *EvoWorkshops* (2005), F. Rothlauf, J. Branke, S. Cagnoni, D. W. Corne, R. Drechsler, Y. Jin, P. Machado, E. Marchiori, J. Romero, G. D. Smith, and G. Squillero, Eds., vol. 3449 of *Lecture Notes in Computer Science*, Springer, pp. 458–467.

[13] Eno, B. *A Year With Swollen Appendices*. Faber & Faber, London, 1996.

[14] Galanter, P. What is generative art? complexity theory as a context for art theory. *proceeding of the Internaional Generative Art conference, Milan* (2003), 216.

[15] Hiller, L., and Issacson, L. *Experimental music : Composition with an Electronic Computer*. McGraw-Hill, New York, 1959.

[16] Holland, J. H. *Emergence, from chaos to order*. Oxford University Press, Oxford, 1998.

[17] Kemp, J., and Howse, M., Eds. *xxxxx*. print on demand, 2007.

[18] Kircher, A. *Musurgia Universalis*. Romæ, 1650.

[19] Latham, R., and Sassen, S., Eds. *Digital Formations*. Princeton University Press, Princeton, N.J., 2005, pp. 178–211.

[20] Laurent, A. M. S. *Understanding Open Source & Free Software Licensing*. O'Reilly Media, Sebastopol, California, August 2004.

[21] Levy, S. *Hackers: heroes of the computer revolution*. Penguin books, London, 1984.

[22] Lovejoy, M. *Digital currents: art in the electronic age*. Routledge, London, 2004.

[23] McDirmid, S. Living it up with a live programming language. In *Proc. Object-Oriented Programming, Systems, Languages Applications 2007* (2007), OOPSLA.

[24] Moody, G. *Rebel Code: Linux and the open source revolution*. Penguin books, London, 2001.

[25] Negroponte, N. *Being Digital*. Hodder & Stoughton, London, 1995.

[26] Nyman, M. *Experimental Music: Cage and Beyond*. Cambridge University Press, Cambridge, 1999.

[27] Partridge, D., and Rowe., J. *Computers and creativity*. Intellect, Oxford, 1994.

[28] Paul, C. *Digital Art*. Thames & Hudson, London, 2003.

[29] Puckette, M. *the Theory and Technique of Electronic Music*. World Scientific Press, Hackensack, N.J., 2007.

[30] Raskin, J. *The Humane Interface*. Addison Wesley, Reading, Mass., 2000.

[31] Raymond, E. S. *The Cathedral & the Bazaar*. O'Reilly, Sebastopol, California, January 2001.

[32] Reynolds, C. W. Flocks, herds, and schools: A distributed behavioral model. In *Proc. ACM SIGRAPH '87* (Anaheim, California, July 1987).

[33] Roads, C., Ed. *Composers and the Computer*. A-R Editions, Middleton, 1985.

[34] Roads, C. *The Music Machine*. MIT Press, Cambridge, Mass, 1989.

[35] Sims, K. Artificial evolution for computer graphics. In *SIGGRAPH* (1991), J. J. Thomas, Ed., ACM, pp. 319–328.

[36] Stallman, R. M. *Free Software Free Society: selected essays of Richard M. Stallman*. GNU press, Boston, MA, 2002.

[37] Todd, S. *Evolutionary Art and Computers*. Academic Press, London, 1992.

[38] Toole, B. A. *Ada, The Enchantress of Numbers*. Strawberry Press, Mill Valley, California, 1992.

[39] Turner, S. R. *The Creative Process, A computer model of storytelling and creativity*. Lawrence Erlbaum Accociates, Hillsdale, N.J., 1994.

[40] Wishart, T. *On Sonic Art*. Harwood Academic, Amsterdam, 1996[1985].

[41] Xenakis, I. *Formalized Music: Thought and Mathematics in Composition.* Indiana University Press, Bloomington, 1971.

[42] Zimmer, F., Ed. *Bang, Pure Data (1. International PD-Convention Graz).* Wolke Verlag, Frankfurt, 2006.

Appendix A

DesireData

URL

http://desiredata.goto10.org

Mailing list

http://lists.goto10.org/cgi-bin/mailman/listinfo/desiredata

IRC

irc.freenode.net, #desiredata

Browse source

http://pure-data.cvs.sourceforge.net/pure-data/pd/src/?pathrev=desiredata

Download Source using CVS

```
cvs     -d:pserver:anonymous@pure-data.cvs.sourceforge.net:/cvsroot/pure-data login
cvs -z6 -d:pserver:anonymous@pure-data.cvs.sourceforge.net:/cvsroot/pure-data co -r desiredata pd
cvs -z6 -d:pserver:anonymous@pure-data.cvs.sourceforge.net:/cvsroot/pure-data co pd/doc
```

Supported operating system

Gnu/Linux

Latest release

http://artengine.ca/desiredata/download/desiredata-2007.08.04.tar.gz

ChangeLog

```
DesireData 2007.08.22 :

 * added more Bokml (Norwegian) translations from Gisle Frysland
 * added Nihongo (Japanese) translations from Kentaro Fukuchi
 * added Dansk (Danish) translations from Steffen Leve Poulsen
 * added History class to unify command history for Listener/Runcommand/TextBox
 * KeyboardDialog clean up, added font selector for console and virtual keyboard
 * Appearance settings can be applied at run time
 * new object/wire indexing system (diff-friendly)
 * Added keyboard/mouse macro recording, playback, and copy (to clipboard)
 * [select] has as many inlets as it has arguments
 * Added [macro] so that a macro can be played back in a patch using messagebox
 * Added [clipboard] to pull the content of system clipboard
 * Fixed variable width font support and TextBox code clean up
 * Added object id display toggle
 * Added [display] object
 * Added patch editing commands
 * Added expand_port
 * Added profiler (object speed measurements) (not compiled in by default)
 * Can now use spaces and \{} in IEM labels and some other places.
 * Added Locale diff tool: localeutils.tcl

DesireData 2007.08.04 :

 * Unicode locales
 * fixed type mismatch bug recently introduced in [unpack]...
```

```
 * fixed lost console posts at startup
 * turned most fprintf() into post() or error()
 * added Chinese locale from Chun Lee
 * added Polish locale from Michal Seta
 * added object creation history
 * added arrow keys and mouse clicks to KeyboardDialog
 * added click drag and copy
 * added background grid
 * added snap to grid
 * added new font selector (in client prefs)

DesireData 2007.07.30 :

 * added classes [unpost], [tracecall], [parse], [unparse]
 * non-constructed objects finally have a dashed box like they used to
 * most of the rest of the C code switched to C++,PD_PLUSPLUS_FACE
 * beginning to use C++ standard library components
 * added event history view (help menu)
 * added keyboard view
 * fixed several bugs in copy/paste, undo/redo, subpatches, gop.
 * added atom_ostream (similar to atom_string)
 * lifted many string length restrictions
 * fixed the hexmunge generator (for classnames with special chars)
 * pd_error() is deprecated
 * added verror(), added open_via_path2(), canvas_open2(), outlet_atom(), ...
 * [route] and [select] support mixed floats and symbols
 * [unpack] supports type "e" meaning any atom ("e" stands for "element")
 * added variable mouse cursor sensitivity
 * various fixes on keyboard navigation

DesireData 2007.06.27 (which should have been 2007.01.12) :

 * merged new loader from Miller's pd 0.40-2
 * merged (but not tested) the rest of the [declare] code from pd 0.40-2
 * added gensym2 (support for NUL in symbols)
 * most of the code now uses C++,PD_PLUSPLUS_FACE,class_new2,etc
```

```
 * auto show/hide scrollbars
 * menu bar can be disabled
 * new Find widget (FireFox style)
 * added "subpatcherize" (turn a selection into a subpatch)
 * IEMGUI can now by controled with keyboard
 * more general keyboard control
 * merged t_alist and t_binbuf together and aliased them to t_list
 * delay uploading until #X restore or #X pop
 * don't upload all abstractions instances to client (much faster)
 * introduced zombie objects to deal with dead objects
 * Command evaluator per canvas window
 * Added locale for Euskara (Basque) by Ibon Rodriguez Garcia
 * PureUnity is now part of the DesireData project (but is designed to
   run also on Miller's 0.40).
 * added -port option in desire.tk so that server and client may
   be started separately.
 * PureUnity has type suffixes for some class families; for each $1 in
   f,~,# (float,signal,grid) there is [inlet.$1] [outlet.$1] [taa.$1]
   [op2.$1] [rand.$1] [norm.$1] [swap.$1] [packunpack3.$1]
 * Other new PureUnity classes: [^] [commutator] [associator]
   [invertor] [distributor] [tree] [protocols-tree]

DesireData 0.40.pre5 (2006.12.19) (-r desiredata; ./configure && make) :

 * merged changes from Miller's pd 0.40-2 (80% of it)
 * new canvas method "reply_with" of canvases replaces the implicit
   reply-matching of pre4. (even less bug-prone)
 * server-side wires and scalars appear to client just like other objects.
 * floatatom,symbolatom,[nbx] use normal Tk text edition just like
   ObjectBox,MessageBox,Comment have done for a while
 * obsolete t_object fields removed: te_type te_width
 * global object table (crash protection for bindless .x targets)
 * variable width font usable in ObjectBoxes and MessageBoxes and Comments.
 * [hsl] [vsl] support jump-on-click again
 * lots of bugfixes
 * -console and -lang moved to Client Preferences dialog
```

```
 * added some more translations by Patrice Colet
 * removed Find menu in main window
 * added Find, Find Next, Find Last Error (canvas windows only)
 * choose between horizontal and vertical in Properties of slider or radio.

DesireData 0.39.A.pre4 (2006.12.07) (-r desiredata; ./configure && make) :

 * major speedup of the GUI (sometimes 3-4 times faster)
 * lots of bugfixes
 * logging of the socket into the terminal is now disabled by default
 * introducing PD_PLUSPLUS_FACE, a new way to use <m_pd.h> and <desire.h>
 * new branch "desiredata" instead of "devel_0_39".
 * got rid of #ifdef DESIRE
 * reply-matching in client-server protocol (less bug-prone)
 * reversing the connection to what it was supposed to be:
   the client connects to the server, not the other way around.
 * the server uses [netreceive] to receive the connection from the GUI
 * removed support for .pdsettings, .plist, microsoft registry.
 * cross-platform libpd
 * new titlebar icon
 * removed t_guiconnect
 * removed [scope]

DesireData 0.39.A.pre3 (2006.11.27) (-r devel_0_39; ./configure && make) :
 * franais updated by Patrice Colet
 * italiano updated by Federico Ferri
 * tons of bugfixes
 * better pdrc editor (renamed to server prefs)
 * removed media menu (split to: help menu, file menu, server prefs)
 * removed Gdb box, added crash report dialog
 * renamed objective.tcl to poe.tcl (because the name was already taken)
 * replaced scons by autoconf and make (starting from Miller's 0.39's files)
 * removed detection of Tcl (we don't need to use libtcl)
 * removed the setuid option because no-one needs it; also fixed the
   setuid security vulnerability in case someone does chmod u+s anyway
 * Portaudio 18 is no longer supported.
```

```
 * simplified configure.in (detector and makefile generator)
 * APIs not compiled in show up in "pd -help", with a special mention
   "(support not compiled in)"; those options don't give you a "unknown
   option" when trying them, it says "option -foo not compiled in this pd".
 * switched desire.c to C++, as another way to reduce redundancy in code.
 * can be compiled without audio support.
 * can be compiled without MIDI support.
 * can --disable-portaudio on OSX
 * added multiple wire connection support
 * fixed copy/paste on canvas
 * keyboard navigation pointer makeover
 * added automatic object insertion support

DesireData 0.39.A.pre2 (2006.11.12) (-r devel_0_39; scons desire=1) :
 * espaol updated by Mario Mora
 * subpatches
 * GOPs
 * abstraction instances
 * multi-line objectboxes, messageboxes and comments
 * keyboard-based navigation
 * made desire.c C++ compatible (for future use)
 * lots of things not written here

DesireData 0.39.A.pre1 (-r devel_0_39; scons desire=1) :
 * merged into the devel branch; enable with scons desire=1, which
   disables lots of g_*.c files (and s_print.c) and enables desire.c;
   use the std devel gui using desire=0.
 * added an object-oriented programming system in desire.tk (do not
   confuse with a dataflow system). added proc unknown, which allows
   subject-verb-complement method-calling in tcl (aka objective.tcl)
 * run the client to start the server and not the other way around: do wish desire.tk
 * the client can make the server start via GDB
 * added Pd box (like Ctrl+M but with history)
 * added Gdb box
 * menu translations in 8 languages
 * classbrowser now show short descriptions in 3 languages
```

```
* objectbox tooltip now replaced by mini-classbrowser
* client conf editor
* other stuff I forget to write about
* looks for .ddrc
* pdrc and ddrc config becomes server and client configuration editor
* graphics rendering completely removed from the server
* toolbar and status bar can be enabled/disabled
* added Patcher->View->Reload: client reloads the patch from the server
* localization support (currently 8 languages: english, franais, deutsch,
    catal, espaol, portugus, bokml, italiano.)
* lots of things not written here
```

Appendix B

List of software

Gnu/Linux

Gnu/Linux is used throughout the research as the operating system. In particular, Gentoo (http://gentoo.org) and pure:dyne (http://puredyne.goto10.org) both have been adopted as the Linux distributions of choice.

Emacs

All text related tasks ranging from programming to writing the thesis are achieved through a extensive text editor called the Emacs
(http://www.gnu.org/software/emacs/).

LaTeX

The thesis employs LaTeX(http://www.latex-project.org/) for formatting and typesetting. LaTeXis widely used in the academic environment to efficiently produce professional documents for publishing.

Tcl/Tk

The client of DesireData is implemented in Tcl/Tk (http://tcl.tk/). Although the DesireData client represents a complete rewrite to Pure Data's GUI, the use of Tcl/Tk is

nevertheless inherited from the existing Pd. Long term plan had been discussed to migrate DesireData client to other languages such as Python (http://www.python.org/).

Pure Data

Pure Data (http://puredata.info) has been used extensively throughout both the personal artistic practice and the research. Refer to the main text (p.98) of the thesis for more detail.

Appendix C

Selected performances and workshops

Selected Performances

- 22/06/2007 **lurk** @ shunt, London, UK
- 03/04/2007 **Make art** @ Confort Moderne, Poitiers, France
- 16/12/2006 **Dorkbot:laptop drumming circle** @ Limehouse Town hall, London, UK
- 03/12/2006 **Elefest/Guerrilla Zoo** @ Corsica Studios, London, UK
- 11/11/2006 **Openlab#3** @ Midnightblue Gallery, London, UK
- 26/10/2006 **Pile on** @ Battersea Barge, London, UK
- 21/10/2006 **Placard** @ state51, London, UK
- 11/10/2006 **Piksel** @ BEK, Bergen, Norway
- 01/09/2006 **Dorkcamp06**, Dorking, UK
- 11/08/2006 **ill FM**, London, UK
- 02/04/2006 **Openlab#2**, @ Vibe Bar, London, UK
- 01/04/2006 **SUM(1,4,6)** @ Area10, London, UK
- 25/03/2006 **Voltage Controlled Cuisine** @ London Social Center, London, UK
- 18/03/2006 **Dorkfest** @ Limehouse Town hall, London, UK
- 27/01/2006 **Make art** @ Confort Moderne, Poitiers, France
- 19/05/2005 **you make me** @ the Bargehouse, London, UK

– 08/04/2005 **Poetic Generative - NEMO festival** @ forum des images, Paris, France

– 01/04/2005 **Openlab #1** @ Foundry, London UK

– 01/02/2005 **Gage05 Festival** @ Hull Time Based Arts, Hull, UK

– 26/03/2004 **Bitsplitters** @ 291 Gallery, London, UK

– 23/08/2003 **Terrain De Jeux 180'** @ Batofar, Paris, France

– 28/06/2003 **Nuit De La Coalition** @ IN FACT, Paris, France

Selected Workshops

– **Pure Data summer school**, Center For Contemporary Art, Glasgow

– **GYOML at the Canteen workshop**, Borrow-in-furness, UK

– **puredata workshop**, spacestudios, Hackney, London

– **Dorkcamp06**, Dorking, UK

– **Pure Data summer school**, Spacestudios, Hackney, London

– **Tagged**, SPACE media, London, UK

– **Goto10 vs Okno**, Brussels, Belgium

Discography

– **Masters of War**, 2004, Incineratemedia
(http://www.incineratemedia.com/masters_of_war/)

– **OutOfChun**, 2007, Catch The Falling Leaves
(http://catchtheleaves.org/)
(http://www.archive.org/details/ctfl004)

Appendix D

CD ROM table of contents

Pure Data patches

- Hypothetical_Waves.pd
- Hypothetical_Waves.mp3

DesireData releases

- **DesireData 2007.08.04.tar.gz**
- **DesireData 2007.07.30.tar.gz**
- **DesireData 2007.06.27.tar.gz**

Music releases

- **Masters of War**
  Assassination Postcard.mp3
  Masters of War.mp3
  Rockets.mp3
  Interval.mp3
  Thanks for Trusting Us.mp3
  MOW 2.0 (Remix by Sonicvariable).mp3
  Mr. President (Remix by Cartesian Lover).mp3
  All of Us Want Peace (Remix by K2).mp3
  AssMastersofWar (Remix by Errorsmith).mp3
  NOTE: Masters of War is a album release between Konrad Kinard (http://incineratemedia.com) and Chun Lee, who used Pure Data in their musical collaboration.

– **OutOfChun**

Glass Cloud.mp3

Gold Spiral.mp3

NEZ.mp3

toy_100p.mp3

i.mp3

NOTE: OutOfChun is a net EP by Chun Lee released on http://catchtheleaves.org/, all music is created using Pure Data.

– **Video documentations**

Cracktux@Shunt.avi

Cracktux@LondonSocialCenter.avi

ClassCloud.avi

NOTE: Cracktux is a performance collaboration between music made by Chun Lee (using Pure Data) and visuals made by Olivier Laruelle (using Processing: http://yesyesnono.co.uk/).